MASTER EN SCIENCES MATHÉMATIQUES

Résolution de moindres carrés linéaires creux avec contraintes de bornes

Bierlaire-Hancotte, Michel

Award date:
1988

Awarding institution:
Universite de Namur

FACULTÉS UNIVERSITAIRES N.D. DE LA PAIX
NAMUR
FACULTÉ DES SCIENCES

Résolution de moindres carrés linéaires creux avec contraintes de bornes

Promoteur : **Ph. TOINT**

Mémoire présenté pour l'obtention du grade
de Licencié en Sciences
mathématiques
par

Michel BIERLAIRE

Facultés Universitaires Notre-Dame de la Paix
Faculté des Sciences
rue de Bruxelles, 61 B-5000 NAMUR
Tél. 081/22.90.61 Télex 59222 facnam-b Téléfax 081/23.03.91

Résolution de moindres carrés linéaires creux avec contraintes de bornes.

BIERLAIRE Michel

Résumé :
Une nouvelle méthode pour détecter l'ensemble actif optimal lorsqu'il s'agit de contraintes de bornes, a été adaptée au problème des moindres carrés linéaires. On l'a comparée ensuite à une méthode de contraintes actives classique.

Abstract :
A new method for detecting the optimal active set has been used to solve the linear least square problem with simple bounds constraints. It is compared with a classic method of active constraints.

Mémoire de licence en Sciences Mathématiques
Juin 1988
Promoteur : **Ph. Toint**

1 Introduction

Le problème des moindres carrés linéaires est un problème de première importance pour beaucoup d'applications (voir définition à la section 2.1).
Supposons par exemple que l'on veuille poser un modèle mathématique linéaire sur des données. Pour réduire le plus possible l'importance des données erronnées, on va utiliser un nombre beaucoup plus grand de mesures qu'il n'y a d'inconnues.
On devra ainsi résoudre un système linéaire surdéterminé, qui est typiquement un problème de moindres carrés.
De plus, dans de nombreux cas, on connaît à l'avance des intervalles où les solutions doivent se trouver (par exemple, lorsqu'il s'agit de probabilités, on est certain qu'elles se situent toujours entre 0 et 1) En imposant aux inconnues de rester dans ces intervalles, on limite encore l'influence des données aberrantes sur la solution.
Concrètement, les procédés d'industrie chimique constituent une partie importante des applications. Notamment, la cohérence des bilans à déjà fait l'objet d'utilisations d'algorithmes de résolution de problèmes des moindres carrés (voir le mémoire de fin d'études de L. Ulrix [14]). Les contraintes de bornes à ce niveau sont essentiellement des contraintes de non négativité, et les matrices sont souvent creuses.
Il y a bien sûr beaucoup d'autres applications en chimie, mais également en physique. Les statisticiens utilisent aussi beaucoup les moindres carrés.

La plupart des algorithmes actuels utilisent, pour résoudre ce genre de problème, des méthodes dites *de contraintes actives.* Nous exposerons à la section 4.2 un algorithme de ce type.
Notre propos est de développer une nouvelle méthode permettant de détecter beaucoup plus rapidement les contraintes qui seront actives à la solution. Cette méthode, qu'appuie une théorie complète décrite par Conn, Gould et Toint dans [2], à déjà été utilisée pour l'optimisation de fonctions non linéaires avec des contraintes de bornes.
Il nous a paru intéressant de l'appliquer au cas particulier de la programmation quadratique, et plus spécialement au problème des moindres carrés.
A la section 2, nous exposerons quelques méthodes, directes et itératives, permettant de résoudre le problème sans contrainte.
Ensuite nous dirons quelques mots de la méthode générale pour l'optimisation non linéaire (section 3).
La section 4 décrira des méthodes de contraintes actives avant de développer la nouvelle méthode.
Enfin, nous donnerons les résultats numériques à la section 5.
Le code des différents programmes qui ont servis aux tests numériques est donné en annexe. Il est écrit en langage FORTRAN, et a été implémenté sur VAX/VMS.

2 Moindres carrés sans contrainte

2.1 Problème

Considérons un système d'équations linéaires

$$Ax = b$$

où

$$A \in \mathbf{R}^{m \times n}$$
$$x \in \mathbf{R}^n$$
$$b \in \mathbf{R}^m$$

Ce système sera appelé *surdéterminé* si $m > n$, c'est-à-dire s'il possède plus d'équations que d'inconnues.
Généralement, ce genre de système ne possède pas de solution exacte. On appellera solution de ce problème le (ou les) x pour le(s)quels Ax sera 'le plus proche possible' de b, i.e.

$$x \quad \text{tq} \quad \forall y \in \mathbf{R}^n \; \|Ax - b\|_p \leq \|Ay - b\|_p$$

la norme traduisant la notion de 'plus proche possible'.

Le problème à résoudre sera donc

$$\min_{x \in \mathbf{R}^n} \|Ax - b\|_p$$

Pour que la fonction à minimiser soit différentiable, nous utiliserons la *norme des moindres carrés* ou *norme 2*.
Le problème des moindres carrés s'énonce donc :

$$\begin{aligned} &\min_{x \in \mathbf{R}^n} \|Ax - b\|_2 \\ &\text{avec} \quad A \in \mathbf{R}^{m \times n} \\ &\qquad\quad b \in \mathbf{R}^m \\ &\qquad\quad m > n \end{aligned} \tag{1}$$

2.2 Définitions

Ensemble des solutions

On définit l'ensemble des solutions du problème (1) comme

$$X = \{x \in \mathbf{R}^n \quad \text{tq} \quad \forall y \in \mathbf{R}^n \; \|Ax - b\|_2 \leq \|Ay - b\|_2\}$$

Résidu

Soit $x \in \mathbf{R}^n$

Le résidu associé au point x est

$$r = b - Ax$$

Equations normales

On appelle *équations normales* le système $n \times n$ d'équations linéaires :

$$A^t A x = A^t b \tag{2}$$

On a

$$\begin{aligned} x \in X \quad & \text{ssi} \quad A^t A x = A^t b \\ & \text{ssi} \quad A^t(b - Ax) = 0 \end{aligned}$$

Cela signifie que le résidu correspondant à une solution est orthogonal aux colonnes de A.

Inverse généralisé

Soit $A \in \mathbf{R}^{m \times n}$

$A^+ \in \mathbf{R}^{n \times m}$ est l'inverse généralisé de A

ssi

A^+ est solution de $\min_{X \in \mathbf{R}^{n \times m}} \|AX - I_m\|_F$

où $\|A\|_F = [\sum_{i=1}^{m} \sum_{i=1}^{n} | a_{ij} |^2]^{\frac{1}{2}}$

$\|.\|_F$ est la norme de Frobenius.

ssi

$$\left.\begin{aligned} AA^+A &= A \\ A^+AA^+ &= A^+ \\ (AA^+)^t &= AA^+ \\ (A^+A)^t &= A^+A \end{aligned}\right\} \text{ Conditions de Moore-Penrose}$$

ssi

AA^+ est la projection orthogonale sur $\mathrm{Im}(A)$

A^+A est la projection orthogonale sur $\mathrm{Im}(A^t)$

Remarques :

- Si rang(A)=n, alors $A^+ = (A^tA)^{-1}A^t$
- Si rang(A)=m=n, alors $A^+ = A^{-1}$

Pour plus de détails sur l'inverse généralisé, vous pouvez consulter [1] et [9].

2.3 Méthodes directes

1. Résolution des équations normales : On a déjà vu que

$$x \in X \iff A^t(b - Ax) = 0_{\mathbf{R}^n}$$

Il suffit donc de résoudre le système linéaire défini par les équations normales pour avoir une solution du problème (1)
Cette méthode nécessite malheureusement le stockage et la factorisation de la matrice A^tA, ce qui n'est pas pratique lorsqu'on traite des problèmes de grande taille et/ou creux.
Nombre d'opérations : $\frac{n^2}{2}(m + \frac{n}{3})$

2. Calcul de l'inverse généralisé

$$x = A^+b$$

Concrètement, cette méthode se ramène à la précédente : on ne calculera jamais A^+ explicitement. On préferera résoudre les équations normales.

3. Décomposition QR :
Soient $A \in \mathbf{R}^{n\times n}$ et $b \in \mathbf{R}^m$.
Soit $Q \in \mathbf{R}^{m\times m}$ orthogonale telle que

$$Q^tA = R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \begin{matrix} n \\ m-n \end{matrix}$$

avec R_1 triangulaire supérieure.

Si

$$Q^tb = \begin{pmatrix} c \\ d \end{pmatrix} \begin{matrix} n \\ m-n \end{matrix}$$

Alors

$$\begin{aligned} \|Ax - b\|_2^2 &= \|Q^tAx - Q^tb\|_2^2 \\ &= \|R_1x - c\|_2^2 + \|d\|_2^2 \end{aligned}$$

Le problème des moindres carrés se ramène alors à un système linéaire triangulaire de dimensions $n \times n$, et donc soluble par substitution.

$$R_1x = c$$

Calcul de la décomposition QR :
Voici deux transformations qui vont permettre la décomposition QR.
Pour plus de détails (algorithmes, démonstrations,..), vous pouvez consulter [1].

(a) *Transformations de Householder*
Soient

$$\begin{aligned} &v \in \mathbf{R}^n \\ &P = I - 2\frac{vv^t}{v^tv} \quad P \in \mathbf{R}^n \end{aligned} \tag{3}$$

P est appelé *réflecteur* ou *transformation de Householder*. L'effet de P sur un vecteur x est de le réfléchir dans l'hyperplan $\text{span}\{v\}^{\perp}$.
Les matrices représentant les transformations de Householder sont symétriques et orthogonales.
De plus, étant donné un vecteur $x \in \mathbf{R}^n, x \neq 0$, on peut construire un vecteur v tel que $Px = (I - 2\frac{vv^t}{v^tv})x$ soit un multiple de e_1 (où e_1 est la première colonne de I_n).

(b) *Transformations de Givens*
Ce sont des corrections de rang 2 de l'identité, de la forme :

$$J(i,k,\theta) = \begin{pmatrix} I & \vdots & & \vdots & \\ \cdots & c & \cdots & s & \cdots \\ & \vdots & I & \vdots & \\ \cdots & -s & \cdots & c & \cdots \\ & \vdots & & \vdots & I \end{pmatrix} \begin{matrix} \\ i \\ \\ k \\ \\ \end{matrix} \tag{4}$$

où $c = \cos\theta$ et $s = \sin\theta$, et I représente une matrice identité de dimensions adéquates.
Il est évident que ces transformations sont orthogonales. En fait, cela revient à faire une rotation dans le plan (i,k) d'un angle θ.
Pour un $x \in \mathbf{R}^n$ donné, on peut choisir θ de manière à ce que le $k^{\text{ième}}$ élément de $J(i,k,\theta)x$ soit nul (avec $i < k$).

La décomposition QR d'une matrice A sera calculée soit en lui appliquant des transformations de Householder judicieusement choisies, soit des transformations de Givens. Comme le produit de matrices orthogonales est orthogonal, on aura bien un facteur Q orthogonal.
Nombre d'opérations :

Décomposition QR :	$n^2(m - \frac{n}{3})$	pour Householder
	$2n^2(m - \frac{n}{3})$	pour Givens
Substitutions :	$\frac{n^2}{2}$	

Inconvénients : si A est creuse, Householder va provoquer du fill-in, et Givens demande deux fois plus d'opérations. (Remarque : il y a moyen d'utiliser une stratégie moins coûteuse, appelée *Givens rapide*, que nous ne développerons pas ici, et qui utilise $n^2(m-\frac{n}{3})$ opérations. Voir [1], [10] et [11])

2.4 Méthode des gradients conjugués

Résoudre un problème des moindres carrés revient à minimiser une fonction quadratique.

$$\begin{aligned} \|Ax - b\|_2^2 &= (Ax-b)^t(Ax-b) \\ &= x^tA^tAx - 2b^tAx + b^tb \end{aligned} \tag{5}$$

Donc, si on ne tient plus compte du terme constant b^tb et qu'on divise cette expression par 2, on obtient une expression équivalente du problème.

$$\begin{gathered} \min_{x\in\mathbf{R}^n} \|Ax - b\|_2^2 \\ \Longleftrightarrow \\ \min_{x\in\mathbf{R}^n} \tfrac{1}{2}x^tA^tAx - b^tAx \end{gathered} \tag{6}$$

Désormais, nous appellerons $Q(x)$ la fonction à minimiser dans le problème équivalent.

$$Q(x) = \frac{1}{2}x^t A^t A x - b^t A x \tag{7}$$

La méthode des gradients conjugués est basée sur le principe suivant :
Pour $k = 1 \ldots n$, on définit x_k comme étant la solution du problème

$$\min_{x \in \text{span}\{p_1 \ldots p_k\}} Q(x)$$

où $p_1 \ldots p_n$ sont les directions de recherche jouissant de la propriété suivante :

$$p_i A^t A p_j = 0 \quad \text{si} \quad i \neq j$$

On dit que les vecteurs p_k sont *conjugués* par rapport à la matrice $A^t A$.
En arithmétique exacte, on aurait que x_n est la solution du problème initial.
Voici l'algorithme théorique proposé par Golub et Van Loan dans [1].

Algorithme 1

Pas 1

$$x_0 = 0$$

Pas 2

Pour $k = 1 \ldots n$ *faire*

Pas 2.1

$$r_{k-1} = A^t b - A^t A x_{k-1}$$
Si $r_{k-1} = 0$ *alors STOP*

Pas 2.2

Si $k = 1$ *alors* $p_k = r_0$
aller en [2.4]

Pas 2.3

$$\beta_k = -\frac{p_{k-1}^t A^t A r_{k-1}}{p_{k-1}^t A^t A p_{k-1}}$$
$$p_k = r_{k-1} + \beta_k p_{k-1}$$

Pas 2.4

$$\alpha_k = \frac{r_{k-1}^t r_{k-1}}{p_k^t A^t A p_k}$$
$$x_k = x_{k-1} + \alpha_k p_k$$

Pas 3

$$x = x_n$$

La référence classique pour la méthode des gradients conjugués est [5]

2.5 Méthode LSQR

Cette méthode a été proposée par Paige et Saunders dans [4]. C'est une méthode itérative pour résoudre des systèmes linéaires ou des moindres carrés de grandes dimensions, et creux. Elle est basée sur deux procédures de bidiagonalisation de la matrice A.

Bidiagonalisation 1

Réduction à une forme bidiagonale inférieure.

Le vecteur de départ est le vecteur b du problème à résoudre.

$$\left.\begin{aligned} \beta_1 u_1 &= b \\ \alpha_1 v_1 &= A^t u_1 \\ \beta_{i+1} u_{i+1} &= A v_i - \alpha_i u_i \\ \alpha_{i+1} v_{i+1} &= A^t u_{i+1} - \beta_{i+1} v_i \end{aligned}\right\} i = 1, 2, \ldots \qquad (8)$$

où $\alpha_i \geq 0$ et $\beta_i \geq 0$ sont choisis tels que $\|u_i\| = \|v_i\| = 1$.

Après k itérations, on obtient les matrices suivantes :

Matrices orthogonales :

$$U_k = [u_1, u_2, \ldots, u_k]$$
$$V_k = [v_1, v_2, \ldots, v_k]$$

En arithmétique exacte, on a que

$$\begin{aligned} U_k^t U_k &= I \\ V_k^t V_k &= I \end{aligned} \qquad (9)$$

Matrice bidiagonale :

$$B_k = \begin{pmatrix} \alpha_1 & & & & & \\ \beta_2 & \alpha_2 & & & & \\ & \beta_3 & \alpha_3 & & & \\ & & \ddots & \ddots & & \\ & & & \ddots & \ddots & \\ & & & & \beta_k & \alpha_k \\ & & & & & \beta_{k+1} \end{pmatrix} \in \mathbf{R}^{(k+1)\times k} \qquad (10)$$

Les relations de récurrence peuvent être écrites sous forme matricielle :

$$\begin{aligned} U_{k+1}(\beta_1 e_1) &= b \\ A V_k &= U_{k+1} B_k \\ A^t U_{k+1} &= V_k B_k^t + \alpha_{k+1} v_{k+1} e_{k+1}^t \end{aligned} \qquad (11)$$

où e_i est un vecteur de longeur k+1 nul partout sauf en $i^{\text{ième}}$ position où il vaut 1

Bidiagonalisation 2

Réduction à une forme bidiagonale supérieure.

Le vecteur de départ est le vecteur $A^t b$.

$$\left.\begin{array}{l} \theta_1 v_1 = A^t b \\ \rho_1 p_1 = A v_1 \\ \theta_{i+1} v_{i+1} = A^t p_i - \rho_i v_i \\ \rho_{i+1} p_{i+1} = A v_{i+1} - \theta_{i+1} p_i \end{array}\right\} i = 1, 2, \ldots \tag{12}$$

où $\theta_i \geq 0$ et $\rho_i \geq 0$ sont choisis tels que $\|p_i\| = \|v_i\| = 1$.
Après k itérations, on obtient les matrices suivantes :
Matrices orthogonales :

$$P_k = [p_1, p_2, \ldots, p_k]$$
$$V_k = [v_1, v_2, \ldots, v_k]$$

En arithmétique exacte, on a que

$$\begin{array}{l} P_k^t P_k = I \\ V_k^t V_k = I \end{array} \tag{13}$$

Matrice bidiagonale :

$$R_k = \begin{pmatrix} \rho_1 & \theta_2 & & & \\ & \rho_2 & \theta_3 & & \\ & & \ddots & \ddots & \\ & & & \ddots & \ddots \\ & & & & \rho_{k-1} & \theta_k \\ & & & & & \rho_k \end{pmatrix} \in \mathbf{R}^{k \times k}$$

Les relations de récurrence peuvent être écrites sous forme matricielle.

$$\begin{array}{lcl} V_k(\theta_1 e_1) & = & A^t b \\ A V_k & = & P_k R_k \\ A^t P_k & = & V_k R_k^t + \theta_{k+1} v_{k+1} e_k^t \end{array} \tag{14}$$

où e_i est un vecteur de longueur k nul partout sauf en $i^{\text{ième}}$ position où il vaut 1

Relations entre les bidiagonalisations

- La matrice V_k est la même dans les deux cas.
- $$B_k^t B_k = R_k^t R_k \tag{15}$$
- Le plus surprenant est que R_k est identique à la matrice obtenue par la factorisation QR de B_k.

$$Q_k B_k = \begin{pmatrix} R_k \\ 0 \end{pmatrix} \text{ avec } Q_k^t Q_k = I \tag{16}$$

Ces identités sont bien sûr valables uniquement en arithmétique exacte.
A partir de la relation (15), on trouve les identités :

$$\begin{aligned}
&\alpha_1^2 + \beta_2^2 = \rho_1^2 \\
&\alpha_i^2 + \beta_{i+1}^2 = \rho_i^2 + \theta_i^2 \quad \text{pour } i > 1 \\
&\alpha_1\beta_1 = \theta_1 \\
&\alpha_i\beta_i = \rho_{i-1}\theta_i \quad \text{pour } i > 1
\end{aligned}$$

Algorithme LSQR :
On veut résoudre le problème

$$\min_x \|Ax - b\|_2$$

Soient :

$$\begin{aligned}
x_k &= V_k y_k \\
r_k &= b - Ax_k \\
t_{k+1} &= \beta_1 e_1 - B_k y_k
\end{aligned}$$

définis à partir d'un vecteur y_k.
En utilisant (8) et (11), on montre que

$$\left.\begin{aligned}
U_{k+1}t_{k+1} &= U_{k+1}\beta_1 e_1 - U_{k+1}B_k y_k \\
&= b - AV_k y_k \\
&= r_k
\end{aligned}\right\} \Rightarrow r_k = U_{k+1}t_{k+1} \qquad (17)$$

Comme U_{k+1} est borné et théoriquement orthonormale, on va choisir y_k de telle manière à minimiser $\|t_{k+1}\|$.
On se ramène donc au problème

$$\min_{y_k} \|\beta_1 e_1 - B_k y_k\|$$

qui est la base du LSQR.
On va résoudre ce problème en utilisant la factorisation QR (16) de B_k. Cela aura la forme :

$$Q_k \begin{bmatrix} B_k & \beta_1 e_1 \end{bmatrix} = \begin{pmatrix} R_k & f_k \\ 0 & \bar{\phi}_{k+1} \end{pmatrix} = \left(\begin{array}{ccccccc|c}
\rho_1 & \theta_2 & & & & & & \phi_1 \\
 & \rho_2 & \theta_3 & & & & & \phi_2 \\
 & & \ddots & \ddots & & & & \vdots \\
 & & & \ddots & \ddots & & & \vdots \\
 & & & & & \rho_{k-1} & \theta_k & \phi_{k-1} \\
 & & & & & & \rho_k & \phi_k \\
\hline
 & & & & & & & \bar{\phi}_{k+1}
\end{array}\right) \qquad (18)$$

où $Q_k = Q_{k,k+1} \ldots Q_{2,3}Q_{1,2}$ est le produit de rotations planes servant à éliminer les éléments sous-diagonaux $\beta_2, \beta_3, \ldots$ de B_k.
On pourra alors calculer y_k et t_{k+1} :

$$\begin{aligned}
&R_k y_k = f_k \\
&t_{k+1} = Q_k^t \begin{pmatrix} 0 \\ \bar{\phi}_{k+1} \end{pmatrix}
\end{aligned} \qquad (19)$$

A partir de là, on peut calculer x_k :

$$\left.\begin{array}{l} x_k = V_k y_k \\ R_k y_k = f_k \end{array}\right\} \Rightarrow x_k = V_k R_k^{-1} f_k = D_k f_k$$

où les colonnes de $D_k = [d_1, d_2, \ldots, d_k]$ peuvent être trouvées successivement par le système :

$$R_k^t D_k^t = V_k^t \tag{20}$$

résolu par substitution.
On aura les formules suivantes :

$$\begin{array}{l} d_k = \frac{1}{\rho_k}(v_k - \theta_k d_{k-1}) \\ x_k = x_{k-1} + \phi_k d_k \end{array} \tag{21}$$

On voit qu'il est nécessaire de stocker uniquement d_{k-1} pour calculer d_k. De plus, on peut épargner une partie du travail en utilisant les vecteurs $w_k = \rho_k d_k$ à la place des d_k.
Remarque : la factorisation QR est déterminée en construisant la $k^{\text{ième}}$ rotation plane $Q_{k,k+1}$ qui agit sur les lignes k et $k+1$ de $[B_k \quad \beta_1 e_1]$ pour éliminer β_{k+1}. On obtient la relation de récurrence :

$$\begin{pmatrix} c_k & s_k \\ s_k & -c_k \end{pmatrix} \begin{pmatrix} \bar{\rho}_k & 0 & \bar{\phi}_k \\ \beta k+1 & \alpha_{k+1} & 0 \end{pmatrix} = \begin{pmatrix} \rho_k & \theta_{k+1} & \phi_k \\ 0 & \bar{\rho}_{k+1} & \bar{\phi}_{k+1} \end{pmatrix} \tag{22}$$

où $\bar{\rho}_1 = \alpha_1$ et $\bar{\phi}_1 = \beta_1$, et c_k et s_k sont les éléments non triviaux de $Q_{k,k+1}$

Il reste à définir des critères d'arrêt :

1. Systèmes compatibles :

   $$\text{Stop si } \|r_k\| \leq BTOL\|b\| + ATOL\|A\|\|x_k\|$$

   où $ATOL$ et $BTOL$ sont spécifiés par l'utilisateur et indiquent la précision sur les données. On peut montrer que, lorsque ce critère est vérifié, x_k est la solution exacte d'un système où A et b ont été perturbés (voir pour cela [4]).

2. Systèmes incompatibles :

   $$\text{Stop si } \frac{\|A^t r_k\|}{\|A\|\|r_k\|} \leq ATOL$$

   Ce critère teste l'orthogonalité entre le résidu et les colonnes de A, en tenant compte des erreurs d'arrondis sur les éléments de A.

3. Systèmes mal conditionnés :

   $$\text{Stop si } \quad \text{cond(A)} \ \geq CONLIM$$

   C'est un critère heuristique détectant un système trop mal conditionné.

Remarque importante :
Si on devait calculer toutes les normes intervenant dans ces critères à chaque itération, ce serait évidemment trop coûteux. Heureusement, il y a moyen de calculer des estimations de ces différentes valeurs à un faible coût.

1. Estimation de $\|r_k\|$
   Par (17) et (19), on a

$$r_k = \bar{\phi}_{k+1} U_{k+1} Q_k^t e_{k+1} \tag{23}$$

   Ainsi, en supposant que $U_{k+1}^t U_{k+1} = I$, on obtient l'estimation :

$$\|r_k\| = \bar{\phi}_{k+1} = \beta_1 s_k s_{k-1} \dots s_1$$

   en se servant de (22) pour calculer $\bar{\phi}_{k+1}$.

2. Estimation de $\|A^t r_k\|$:
   Par (23),(11) et (18),

$$\begin{aligned} A^t r_k &= \bar{\phi}_{k+1}(V_k B_k^t + \alpha_{k+1} v_{k+1} e_{k+1}^t) Q_k^t e_{k+1} \\ &= \bar{\phi}_{k+1} V_k [R_k^t \;\; 0] e_{k+1} + \bar{\phi}_{k+1} \alpha_{k+1} (e_{k+1}^t Q_k^t e_{k+1}) v_{k+1} \end{aligned}$$

   Le premier terme disparaît et on peut voir que le $(k+1)^{\text{ième}}$ terme de la diagonale de Q_k est $-c_k$. On a donc :

$$A^t r_k = -(\bar{\phi}_{k+1} \alpha_{k+1} c_k) v_{k+1}$$

   et

$$\|A^t r_k\| = \bar{\phi}_{k+1} \alpha_{k+1} |c_k|$$

   Ici, il n'y a aucune hypothèse d'orthogonalité.

3. Estimation de $\|x_k\|$:
   La matrice bidiagonale supérieure R_k peut être réduite à une forme bidiagonale inférieure par la factorisation orthogonale

$$R_k \bar{Q}_k^t = L_k$$

   où $\bar{Q}_k$ est un produit de rotations planes.
   Si on définit $\bar{z}_k$ par le système

$$L_k \bar{z}_k = f_k$$

   il s'ensuit que

$$x_k = (V_k R_k^{-1}) f_k = (V_k \bar{Q}_k^t) \bar{z}_k \equiv \bar{W}_k \bar{z}_k$$

   En supposant que $V_k^t V_k = I$, on obtient

$$\|x_k\| = \|\bar{z}_k\|$$

   On peut ainsi estimer $\|x_k\|$ en 13 multiplications (voir [4]), ce qui est négligeable pour de grandes dimensions.

4. Estimation de $\|A\|_F$ et cond(A) :
   Par (8), il est évident que les v_i appartiennent à $\mathrm{Im}(A^t)$, et sont donc orthogonaux au noyau de A et à celui de $A^t A$. Ainsi, en supposant les propriétés d'orthogonalité et en utilisant (11), on a

$$B_k^t B_k = V_k^t A^t A V_k$$

Par le théorème du "minimax" de Courant-Fisher (voir [12]), les valeurs propres de $B_k^t B_k$ sont entrelacées avec celles de A^tA et sont bornées supérieurement et inférieurement par la plus grande et la plus petite (resp.) valeur propre non nulle de A^tA. On peut évidemment dire la même chose pour les valeurs singulières de B_k comparées à celles de A. Ainsi, pour la norme 2 et la norme F, on a

$$\|B_k\| \leq \|A\|$$

Nous allons utiliser $\|B_k\|_F$ comme une estimation de la norme de A croissant de façon monotone.
On peut montrer de manière analogue que

$$\|R_k^{-1}\| = \|B_k^+\| \leq \|A^+\|$$

Comme $D_k = V_k R_k^{-1}$, on a

$$1 \leq \|B_k\|\|D_k\| \leq \|A\|\|A^+\| = \text{ cond(A)}$$

$\|B_k\|_F\|D_k\|_F$ sera pris comme estimation (croissant de façon monotone au cours des itérations) de cond(A).
Le coût de cette estimation n'est pas très élevé :

$$\|B_k\|_F^2 = \|B_{k-1}\|_F^2 + \alpha_k^2 + \beta_{k+1}^2$$
$$\|D_k\|_F^2 = \|D_{k-1}\|_F^2 + \|d_k\|^2$$

On peut maintenant décrire l'algorithme (dans lequel on ne développera pas le calcul des différentes estimations des normes).

Algorithme 2

1. **Initialisations**

$$\beta_1 u_1 = b$$
$$\alpha_1 v_1 = A^t u_1$$
$$w_1 = v_1 \qquad (= \rho_1 d_1)$$
$$x_0 = 0$$
$$\bar{\phi}_1 = \beta_1$$
$$\bar{rho}_1 = \alpha_1$$

2. **Pour i=1,2,3,... répéter les pas 2a à 2d**

(a) Continuer la bidiagonalisation :

$$\left.\begin{aligned} \beta_{i+1} u_{i+1} &= Av_i - \alpha_i u_i \\ \alpha_{i+1} v_{i+1} &= A^t u_{i+1} - \beta_{i+1} v_i \end{aligned}\right\} \textit{ voir (8)}$$

(b) Construire et appliquer la transformation orthogonale suivante :

$$\begin{aligned}
\rho_i &= (\bar{\rho}_i^2 + \beta_{i+1}^2)^{\frac{1}{2}} \\
c_i &= \frac{\bar{\rho}_i}{\rho_i} \\
s_i &= \frac{\beta_{i+1}}{\rho_i} \\
\theta_{i+1} &= s_i \alpha_{i+1} \\
\bar{\rho}_{i+1} &= -c_i \alpha_{i+1} \\
\phi_i &= c_i \bar{\phi}_i \\
\bar{\phi}_{i+1} &= s_i \bar{\phi}_i
\end{aligned}$$

(c) Mise à jour de x et w :

$$\begin{aligned}
x_i &= x_{i-1} + \left(\frac{\phi_i}{\rho_i}\right) w_i \\
w_{i+1} &= v_{i+1} - \left(\frac{\theta_{i+1}}{\rho_i}\right) w_i
\end{aligned}$$

(d) Tests de convergence :
Les critères d'arrêt sont formulés en termes de 3 quantités sans dimension : ATOL, BTOL et CONLIM, qui seront spécifiées par l'utilisateur.

S1 *Stop si*

$$\|r_k\| \leq BTOL\|b\| + ATOL\|A\|\|x_k\|$$

S2 *Stop si*

$$\frac{\|A^t r_k\|}{\|A\|\|r_k\|} \leq ATOL$$

S3 *Stop si*

$$cond(A) \geq CONLIM$$

2.6 Liens entre LSQR et les gradients conjugués

Théorème 3
Les vecteurs de recherche de LSQR sont conjugués par rapport à A^tA.
Démonstration.
Par (14) et (13), on a :

$$\begin{aligned}
AV_k &= P_k R_k \\
A^t P_k &= V_k R_k^t + \theta_{k+1} v_{k+1} e_k^t
\end{aligned}$$

et donc

$$\begin{aligned}
V_k^t A^t A V_k &= V_k^t A^t P_k R_k \\
&= V_k^t V_k R_k^t R_k + \theta_{k+1} V_k^t v_{k+1} e_k^t R_k \\
&= R_k^t R_k
\end{aligned}$$

Par (20) on a que :

$$\begin{aligned}
V_k &= D_k R_k \\
R_k^t D_k^t A^t A D_k R_k &= R_k^t R_k
\end{aligned}$$

ce qui donne

$$D_k^t A^t A D_k = I \tag{24}$$

soit encore

$$d_i^t A^t A d_j = \begin{cases} 0 & si\ i \neq j \\ 1 & si\ i = j \end{cases}$$

Les vecteurs d_k sont donc bien conjugués par rapport à A^tA. □

Théorème 4
La première itération est la même pour LSQR et les gradients conjugués.
Démonstration.

1. *LSQR*

- *Première direction de recherche :*
  Dans les initialisations de l'algorithme 2, on a que

$$d_1 = \frac{1}{\rho_1} v_1$$

  Et par (12), on obtient

$$d_1 = \frac{1}{\rho_1 \theta_1} A^t b \tag{25}$$

  Cette direction est celle du gradient en $x_0 = 0$

- *Pas le long de cette direction :*
  (voir les relations de récurrence de l'algorithme 2 décrit à la section 2.5)

$$\begin{aligned} x_1 &= x_0 + \phi_1 d_1 \\ &= \tfrac{\phi_1}{\rho_1 \theta_1} A^t b \end{aligned} \tag{26}$$

$$\begin{aligned} \phi_1 &= c_1 \beta_1 \\ c_1 &= \tfrac{\bar{\rho}_1}{\rho_1} \\ &= \tfrac{\alpha_1}{\rho_1} \end{aligned}$$

  Ainsi

$$\phi_1 = \frac{\alpha_1 \beta_1}{\rho_1} \tag{27}$$

$$\alpha_1 v_1 = A^t u_1 = \frac{1}{\beta_1} A^t b$$

  ce qui donne

$$\alpha_1 = \frac{1}{\beta_1} \|A^t b\|$$

  et

$$\|A^t b\| = \alpha_1 \beta_1 \tag{28}$$

  Par les relations de récurrence,

$$\theta_1 v_1 = A^t b$$

  Comme v_1 est normalisé

$$\theta_1 = \|A^t b\| \tag{29}$$

Par (28) on a

$$\theta_1 = \alpha_1 \beta_1$$

et donc

$$\phi_1 = \frac{\theta_1}{\rho_1}$$

Le pas vaut donc :

$$\frac{\phi_1}{\rho_1 \theta_1} = \frac{\theta_1}{\rho_1^2 \theta_1} = \frac{1}{\rho_1^2}$$

Or,

$$rho_1 p_1 = Av_1$$

Comme p_1 est normalisé

$$\rho_1 = \|Av_1\|$$

où $v_1 = \frac{1}{\theta_1} A^t b$. Donc,

$$\rho_1^2 = v_1^t A^t A v_1 = \frac{1}{\theta_1^2} b^t A A^t A A^t b$$

et par (29) le pas vaut :

$$\frac{1}{\rho_1^2} = \frac{b^t A A^t b}{b^t A A^t A A^t b} \tag{30}$$

2. Gradients conjugués :
(voir l'algorithme 1 à la section 2.4)

- *Première direction de recherche :*

$$p_1 = r_0 = A^t b - A^t A x_0 = A^t b$$

- *Pas effectué le long de cette direction :*

$$\frac{r_o^t r_o}{p_1^t A^t A p_1} = \frac{p_1^t p_1}{p_1^t A^t A p_1}$$

On peut faire le rapprochement avec LSQR :

$$\begin{aligned} p_1 &= A^t b \\ d_1 &= \frac{1}{\rho_1 \theta_1} \|A^t b\| \end{aligned}$$

et on obtient le rapprt entre les deux directions:

$$p_1 = \rho_1 \theta_1 d_1$$

où p_1 est la première direction de recherche des gradients conjugués, et d_1 celle du LSQR. Donc, par (24), le pas vaut

$$\frac{(\rho_1 \theta_1)^2 d_1^t d_1}{(\rho_1 \theta_1)^2 d_1^t A^t A d_1} = d_1^t d_1$$

Or,

$$d_1 = \frac{1}{\rho_1 \theta_1} A^t b$$

et donc par (29) et (30), le pas vaut :

$$\frac{1}{\rho_1^2\theta_1^2}b^tAA^tb = \frac{1}{\rho_1^2} = \frac{b^tAA^tb}{b^tAA^tAA^tb}$$

On obtient donc la même direction de recherche et le même pas pour LSQR que pour les gradients conjugués.

□

En fait, ce pas est le résultat d'une recherche linéaire exacte le long de la plus forte pente en $x_0 = 0$.
En effet, on a

$$Q(x) = \frac{1}{2}x^tA^tAx - b^tAx$$

Si on pose $x_1 = x_0 + \alpha A^tb$, on obtient

$$\begin{aligned} Q(x_1) &= Q(x_0 + \alpha A^tb) \\ &= Q(\alpha A^tb) \\ &= \tfrac{1}{2}\alpha^2 b^tAA^tAA^tb - \alpha b^tAA^tb \end{aligned}$$

et donc,

$$\frac{d}{d\alpha}Q(x_1) = \alpha b^tAA^tAA^tb - b^tAA^tb$$

et

$$\frac{d}{d\alpha}Q(x_1) = 0$$

ce qui donne

$$\alpha = \frac{b^tAA^tb}{b^tAA^tAA^tb}$$

Théorème 5
LSQR est une méthode de descente.
Démonstration.
Il faut montrer que la valeur de la quadratique (5) diminue à chaque itération.

- *Conditions générales :*
  On a

$$\begin{aligned} Q : \ \mathbf{R}^n &\longrightarrow \mathbf{R} \\ x &\rightsquigarrow \tfrac{1}{2}x^tA^tAx - x^tA^tb \end{aligned}$$

  Soit $x_{k-1} \in \mathbf{R}^n$ quelconque.
  Soit $x_k = x_{k-1} + \alpha_k p_k$
  où
  - *$p_k \in \mathbf{R}^n$ est une direction de recherche*
  - *$\alpha_k \in \mathbf{R}$ est la longueur du pas effectué le long de cette direction.*

Essayons de trouver les conditions pour que

$$Q(x_k) < Q(x_{k-1})$$

Développons l'expression de $Q(x_k)$:

$$\begin{aligned} Q(x_k) &= Q(x_{k-1} + \alpha_k p_k) \\ &= \tfrac{1}{2}(x_{k-1} + \alpha_k p_k)^t A^t A(x_{k-1} + \alpha_k p_k) - (x_{k-1} + \alpha_k p_k)^t A^t b \\ &= \tfrac{1}{2} x_{k-1}^t A^t A x_{k-1} + \tfrac{1}{2}\alpha_k^2 p_k^t A^t A p_k + \alpha_k p_k^t A^t A x_{k-1} - x_{k-1}^t A^t b - \alpha_k p_k^t A^t b \\ &= Q(x_{k-1}) + \tfrac{1}{2}\alpha_k^2 p_k^t A^t A p_k + \alpha_k (p_k^t A^t A x_{k-1} - p_k^t A^t b) \end{aligned}$$

Si on définit :

$$r_{k-1} = A^t b - A^t A x_{k-1} \quad \in \mathbf{R}^n$$

on a

$$Q(x_k) = Q(x_{k-1}) + \frac{1}{2}\alpha_k^2 p_k^t A^t A p_k - \alpha_k p_k^t r_{k-1}$$

Supposons que p_k *soit connue, et cherchons le* α_k *qui minimise* $Q(x_k)$.
On va donc annuler la dérivée de $Q(x_k)$ *par rapport à* α_k.

$$\begin{aligned} \tfrac{d}{d\alpha_k} Q(x_k) &= 0 \\ \alpha_k p_k^t A^t A p_k - p_k^t r_{k-1} &= 0 \end{aligned}$$

ce qui donne

$$\alpha_k = \frac{p_k^t r_{k-1}}{p_k^t A^t A p_k}$$

On aura donc :

$$\begin{aligned} Q(x_k) &= Q(x_{k-1}) + \tfrac{1}{2}\left(\tfrac{p_k^t r_{k-1}}{p_k^t A^t A p_k}\right)^2 p_k^t A^t A p_k - \tfrac{p_k^t r_{k-1}}{p_k^t A^t A p_k} p_k^t r_{k-1} \\ &= Q(x_{k-1}) - \tfrac{1}{2}\tfrac{(p_k^t r_{k-1})^2}{\|A p_k\|^2} \end{aligned}$$

On voit que, pour assurer une diminution de Q, *il faut que*

$$p_k^t r_{k-1} \neq 0$$

- *Dans le cas de LSQR, il va falloir montrer que :*
  1. *La direction de recherche* d_k *n'est pas orthogonale à* $r_{k-1} = A^t b - A^t A x_{k-1}$
  2. *Le pas effectué le long de cette direction est bien*

$$\alpha_k = \frac{d_k^t r_{k-1}}{d_k^t A^t A d_k}$$

1. *Montrons que la direction de recherche d_k n'est pas orthogonale à $r_{k-1} = A^t b - A^t A x_{k-1}$*

$$d_k^t r_{k-1} = d_k^t A^t b - d_k^t A^t A x_{k-1} \tag{31}$$

Or, par sa construction même au cours de l'algorithme, on sait que

$$x_{k-1} \in \text{span } \{d_1, \ldots, d_{k-1}\} \tag{32}$$

Par le théorème 3 et (32), on a que

$$d_k^t A^t A x_{k-1} = 0 \tag{33}$$

Ainsi, en remplacant (33) dans (31) :

$$d_k^t r_{k-1} = d_k^t A^t b \tag{34}$$

Il reste à montrer que $d_k^t A^t b \neq 0$, ce que l'on va faire par récurrence.

$k = 1$ Par (25),

$$d_1 = \frac{A^t b}{\theta_1 \rho_1}$$

et donc

$$d_1^t A^t b = \frac{1}{\theta_1 \rho_1} \|A^t b\|^2 \neq 0$$

$k > 1$ Supposons que

$$d_{k-1}^t A^t b \neq 0 \tag{35}$$

Par (12) on a

$$A^t b = \theta_1 v_1$$

Comme les vecteurs v_k sont orthogonaux, on a

$$v_k^t A^t b = 0 \tag{36}$$

Par (21) et (36) on a

$$\begin{aligned} d_k^t A^t b &= \tfrac{1}{\rho_k}(v_k^t A^t b - \theta_k d_{k-1}^t A^t b) \\ &= -\tfrac{\theta_k}{\rho_k} d_{k-1}^t A^t b \end{aligned}$$

Par (35), on a :

$$\begin{aligned} &\forall k > 1 \\ &\textit{Si } \theta_k = 0 \quad \textit{alors} \quad Q(x_k) = Q(x_{k-1}) \\ &\textit{Si } \theta_k \neq 0 \quad \textit{alors} \quad Q(x_k) < Q(x_{k-1}) \end{aligned}$$

2. *Montrons que le pas effectué le long de la direction de recherche est bien*

$$\alpha_k = \frac{d_k^t r_{k-1}}{d_k^t A^t A d_k} \tag{37}$$

Par (24) et (34) :

$$\alpha_k = \frac{d_k^t r_{k-1}}{d_k^t A^t A d_k} = d_k^t A^t b$$

Le pas effectué par LSQR le long de d_k est ϕ_k. Montrons par récurrence que

$$\phi_k = d_k^t A^t b$$

$k = 1$ *Par (29),(28) et (25) :*

$$\begin{aligned}\phi_1 &= \frac{\theta_1}{\rho_1} \\ &= \frac{\theta_1^2}{\rho_1\theta_1} \\ &= \frac{1}{\rho_1\theta_1}\|A^t b\|^2 \\ &= \frac{1}{\rho_1\theta_1} b^t A A^t b \\ &= d_1^t A^t b\end{aligned}$$

$k > 1$ *Supposons*

$$\phi_{k-1} = d_{k-1}^t A^t b \tag{38}$$

et montrons

$$\phi_k = d_k^t A^t b$$

Rappelons d'abord la définition de r_k dans LSQR :

$$r_k = b - Ax_k$$

Par (17) et (19)

$$\begin{aligned}r_k &= U_{k+1} t_{k+1} \\ t_{k+1} &= Q_k^t \begin{pmatrix} 0_k \\ \phi_{k+1} \end{pmatrix} = Q_k^t \phi_{k+1} e_{k+1}\end{aligned}$$

et on obtient

$$r_k = \phi_{k+1} U_{k+1} Q_k^t e_{k+1} \tag{39}$$

Par (39),la décomposition QR (16), et les relations de récurrence (11) et (22),

$$\begin{aligned}A^t r_k &= \phi_{k+1} A^t U_{k+1} Q_k^t e_{k+1} \\ &= \phi_{k+1} V_k B_k^t Q_k^t e_{k+1} + \phi_{k+1}\alpha_{k+1} v_{k+1}(e_{k+1}^t Q_k^t e_{k+1}) \\ &= \phi_{k+1} V_k \underbrace{[R_k \;\; 0] e_{k+1}}_{=0} - \phi_{k+1}\alpha_{k+1} c_k v_{k+1} \\ &= -\phi_{k+1}\alpha_{k+1} c_k v_{k+1}\end{aligned} \tag{40}$$

Reprenons à partir de r_k :

$$\begin{aligned}r_k &= b - Ax_k \\ A^t r_k &= A^t b - A^t A x_k \\ d_k^t A^t r_k &= d_k^t A^t b - d_k^t A^t A(x_{k-1} + \phi_k d_k) \\ &= d_k^t A^t b - \phi_k\end{aligned} \tag{41}$$

On doit donc parvenir à montrer que

$$d_k^t A^t r_k = 0$$

C'est bien le cas, car, par (21) :

$$d_k \in \text{span}\,\{v_1, \ldots, v_k\}$$

Comme les v_k sont orthogonaux, on a que

$$d_k^t v_{k+1} = 0$$

Il reste à utiliser (40) pour terminer la démonstration. En effet, on a

$$d_k^t v_{k+1} = 0$$

et donc

$$d_k^t A^t r_k = 0$$

Par (41), on a donc

$$\phi_k = d_k^t A^t b$$

ce qu'il fallait démontrer.

On a ainsi vérifié qu'une itération de LSQR diminuait la valeur du résidu. □

Essayons maintenant de prouver l'affirmation de Paige et Saunders comme quoi l'algorithme LSQR génère les mêmes points que l'algorithme des gradients conjugués.
Nous avons déjà montré par le théorème 4 que la première direction de recherche est la même pour les deux méthodes. Nous allons supposer qu'il en est de même pour les directions de recherche suivantes.

Théorème 6
En supposant que les directions de recherche sont les mêmes, LSQR génère les mêmes points que l'algorithme des gradients conjugués.
Démonstration.
Il suffit de montrer que, dans l'algorithme des gradients conjugués, le pas effectué est bien :

$$\alpha_k = \frac{p_k^t r_{k-1}}{p_k^t A^t A p_k}$$

En effet, nous avons déjà montré au thérème 5 que c'était bien le pas effectué par LSQR.
Le pas effectué par les gradients conjugués est

$$\alpha_k = \frac{r_{k-1}^t r_{k-1}}{p_k^t A^t A p_k}$$

Il suffit donc de montrer que

$$p_k^t r_{k-1} = r_{k-1}^t r_{k-1}$$

On peut montrer que (voir Golub et Van Loan [1], p. 368)

$$p_k \in \ span\{r_{k-1}, p_{k-1}\}$$

Sans perte de généralité, on peut supposer que

$$p_k = r_{k-1} + \beta_k p_{k-1}$$

Ainsi

$$p_k^t r_{k-1} = r_{k-1}^t r_{k-1} + \beta_k p_{k-1}^t r_{k-1}$$

Il reste à montrer que

$$p_{k-1}^t r_{k-1} = 0$$

Comme

$$\begin{aligned} p_{k-1} &= r_{k-2} + \beta_{k-1} p_{k-2} \\ p_{k-2} &= r_{k-3} + \beta_{k-2} p_{k-3} \\ &etc... \\ p_1 &= r_0 \end{aligned}$$

on a que

$$p_{k-1} \in \ span\{r_0, \ldots, r_{k-2}\}$$

Comme les r_k sont orthogonaux entre eux (voir Golub et Van Loan [1] p. 368), on a bien que

$$p_{k-1}^t r_{k-1} = 0$$

□

3 Optimisation non linéaire avec contraintes de bornes

3.1 Enoncé du problème

$$\begin{cases} \min_{x \in \mathbf{R}^n} f(x) \\ \text{s.c. } l_i \leq x_i \leq u_i (i = 1, \ldots, n) \end{cases} \tag{42}$$

où $f : \mathbf{R}^n \to \mathbf{R}$

3.2 Type de méthode utilisée

Dans le cas où f est une fonction non linéaire[1] générale, on applique un algorithme du type région de confiance :
A chaque itération, on définit une approximation quadratique de la fonction objectif, et une région autour de l'itéré courant où l'on croit cette approximation bonne. Ensuite, on calcule dans cette région un candidat pour l'itéré suivant qui réduit suffisamment la valeur de l'approximation quadratique (appelée *modèle de la fonction*). Si la valeur de la fonction en ce point se rapproche suffisamment de la valeur prévue, le nouveau point est accepté comme itéré suivant et la région de confiance éventuellement élargie. Autrement, le point est rejeté et on diminue la région de confiance.
Nous ne donnerons pas plus de détails sur les régions de confiance pour la bonne raison qu'elles ne nous intéressent pas dans le cadre de notre problème : en effet, la fonction objectif est quadratique (voir équation (7)). L'approximation sera donc partout égale à la fonction, et la région de confiance sera bien sûr $\mathbf{R}^n$ tout entier.
Pour information, voici, en résumé, la méthode telle qu'elle a été proposée par Conn, Gould et

[1] deux fois différentiable

Toint dans [2] et [3]. Nous verrons après comment l'adapter au problème qui nous préoccupe.

3.3 Définitions

Commençons par définir quelques concepts.

1. **Région admissible :**
   On définit la région admissible $\mathcal{F}$ comme :

$$\mathcal{F} = \{x \in \mathbf{R}^n \text{ t.q. } \forall i \in \{1, \ldots, n\} \quad l_i \leq x_i \leq u_i\} \tag{43}$$

2. **Ensemble actif :**
   L'ensemble actif associé à un point x est un ensemble d'indices défini comme suit :

$$I(x) = \{i \text{ t.q. } x_i \leq l_i \text{ ou } x_i \geq u_i\} \tag{44}$$

3. **Projection sur la région admissible :**
   On définit l'opérateur de projection P composante par composante

$$P[x]_i = \begin{cases} l_i & \text{si } x_i \leq l_i \\ u_i & \text{si } x_i \geq u_i \\ x_i & \text{sinon} \end{cases} \tag{45}$$

4. **Modèle quadratique :**
   A la $k^{\text{ième}}$ itération, on a :

   - x_k un point admissible
   - g_k le gradient de f en x_k
   - B_k une bonne approximation symétrique du hessien en ce point.

   Le modèle quadratique est défini comme suit :

$$m_k(x_k + s) = f(x_k) + g_k^t s + \frac{1}{2} s^t B_k s \tag{46}$$

5. **Région de confiance admissible :**
   C'est simplement l'intersection de la région de confiance et de la région admissible. Il est à remarquer que dans le problème de moindres carrés linéaires, cette région coïncide avec la région admissible.
   On peut définir un opérateur de projection P' sur cette région de la même manière qu'en (45).

6. **Problème de la région de confiance :**

$$\begin{aligned} &\min_{s \in \mathbf{R}^n} m_k(x) \\ &\text{s.c } l \leq x \leq u \\ &\text{et } \|x - x_k\| \leq \Delta_k \end{aligned}$$

où $m_k(x)$ est défini en (46), Δ_k est le rayon de la région de confiance et $\|.\|$ est une norme convenablement choisie. Dans ce cas-ci, on prendra de préférence la norme ∞. Ce choix est dû à la forme de la région admissible : les contraintes de bornes définissent un domaine qui a la forme d'un pavé. L'intersection entre la région de confiance et ce domaine sera plus facilement déterminée si cette région de confiance a également la forme d'un pavé. Dorénavant, pour ne pas compliquer inutilement les explications, nous appellerons cette intersection la région admissible.

7. **Point de Cauchy généralisé (PCG):**
   Le point de Cauchy généralisé est défini comme le premier minimum local de la fonction à une variable :

$$q_k(t) \stackrel{\text{def}}{=} m_k(P[x_k - tg_k])$$

   i.e. le premier minimum local du modèle le long de la ligne brisée obtenue en projetant la direction de plus forte pente $(-g_k)$ sur la région admissible.

3.4 Calcul du point de Cauchy généralisé

Pour trouver le point de Cauchy généralisé, nous allons considérer l'un après l'autre les arcs situés entre deux points de cassure consécutifs (un point de cassure étant un point où une des variables atteint une de ses bornes).
Voici en gros la méthode utilisée.

Initialisations
D'abord, on définit la première direction de recherche. Pour cela, on calcule l'ensemble actif de l'itéré courant. La direction de recherche d est définie composante par composante.

$$d_i = \begin{cases} 0 & \text{si } i \in I(x_k) \\ -(g_k)_i & \text{sinon} \end{cases}$$

On calcule les dérivées directionnelles du modèle le long de cette direction :

$$f' = g_k^t d$$
$$f'' = d^t B_k d$$

On peut déjà arrêter si la dérivée première est positive ou nulle, car alors, la valeur de la quadratique augmente le long de la direction.

Itérations :

- On détermine le point de cassure suivant, c'est-à-dire qu'on calcule le pas (λ_k) le plus grand que l'on puisse faire le long de la direction sans sortir de la région de confiance admissible (définie en 5, section 3.3). On calcule l'ensemble actif en ce point.

- On regarde alors si le point de Cauchy généralisé se trouve sur l'arc ainsi défini. Pour cela, on calcule le minimum local le long de la direction courante. Il y a deux solutions possibles :

  1. Le minimum calculé se trouve sur l'arc. Dans ce cas on a trouvé de PCG.
  2. Le minimum calculé ne se trouve pas sur l'arc et on passe à l'arc suivant : on vérifie d'abord que la fonction descend encore après le point de cassure (dans le cas contraire ce dernier serait le PCG) et on recommence en suivant une nouvelle direction $\bar{d}_k$ t.q.

$$(\bar{d}_k)_i = \begin{cases} 0 & \text{si } i \in I(\bar{x}_k) \text{ et } i \notin I(x_k) \\ (d_k)_i & \text{sinon} \end{cases}$$

  avec $\bar{x}_k = x_k + \lambda_k d_k$

3.5 Algorithme

Voici maintenant l'algorithme tel qu'il a été proposé par Conn, Gould et Toint dans [3].

Pas 0 : *Initialisations*

- Un point de départ admissible : x_0
- La valeur de la fonction : $f(x_0)$
- La valeur du gradient en x_0 : g_0
- Le rayon de la région de confiance initiale : Δ_0
- Une approximation symétrique de la matrice hessienne en x_0 : B_0
- $k = 0$

Pas 1 : *Test de convergence*

On calcule le gradient projeté :

$$g_k^p = P[x_k - g_k] - x_k \tag{47}$$

où l'opérateur P est celui défini en (45).
On estime avoir trouvé la solution si $\|g_k^p\| < \varepsilon$ pour un ε assez petit donné par l'utilisateur.
STOP.

Pas 2 : *Calcul du point de Cauchy généralisé*

Voici le détail de la méthode expliquée plus haut.

Pas 2.0 : *Initialisations*

$$\begin{aligned} x &= x_k \\ g &= g_k - B_k x_k \\ d_i &= \begin{cases} 0 & \text{si } i \in I(x_k) \\ -(g_k)_i & \text{sinon} \end{cases} \quad \text{pour } i = 1, \ldots, n \\ f' &= g_k^t d \\ f'' &= d^t B_k d \end{aligned}$$

Si $f' \geq 0$ aller en *2.4*

Pas 2.1 : *Trouver le point de cassure suivant*

Pour chaque i on défini :

$$\Delta t_i = \begin{cases} \frac{u_i - x_i}{d_i} & \text{si } d_i > 0 \\ \frac{l_i - x_i}{d_i} & \text{si } d_i < 0 \\ \infty & \text{si } d_i = 0 \end{cases}$$

Le pas recherché est

$$\Delta t = \min_{i=1,\ldots,n} \Delta t_i$$

et le nouveau point de cassure sera

$$x + \Delta t\, d$$

Calculer l'ensemble J des indices des variables qui rencontrent pour la première fois leur borne en ce point.

Pas 2.2 : *Voir si on a trouvé le PCG*

Si $f'' > 0$ (pour avoir un minimum et non un maximum) et $0 < -\frac{f'}{f''} < \Delta t$ (le minimum local se trouve avant le point de cassure), alors

$$x \leftarrow x - \frac{f'}{f''} d$$

Aller en **2.4**

Pas 2.3 : *Mises à jour*

$$\begin{aligned} b &= B_k\left(\textstyle\sum_{i \in J} d_i e^i\right) \\ x &= x + \Delta t\, d \\ f' &= f' + \Delta t\, f'' - b^t x - \textstyle\sum_{i \in J} d_i g_i \\ f'' &= f'' + b^t\left(\textstyle\sum_{i \in J} d_i e^i - 2d\right) \\ d_i &= 0 \quad \forall i \in J \end{aligned}$$

Si $f' \geq 0$ aller en **2.4** , sinon aller en **2.1**.

Pas 2.4 : *On a trouvé le PCG*

$$x_k^c = x$$

Pas 3 : *Trouver le nouvel itéré*

On fixe les variables actives au PCG, et on applique un algorithme de gradients conjugués au sous-espace des variables non fixées. On aura :

$$(x_{k+1})_i = \begin{cases} (x^c)_i & \text{si } i \text{ est active} \\ (x^g)_i & \text{sinon} \end{cases}$$

où x^g est la solution trouvée par la méthode des gradients conjugués dans le sous-espace.

Pas 4 : *Mises à jour*

On vérifie d'abord si le point ainsi déterminé est acceptable.

- Si c'est le cas, on fait la mise à jour du rayon de la région de confiance et de l'approximation de la matrice hessienne.
- Sinon, on pose $x_{k+1} = x_k$ et on réduit le rayon de la région de confiance.

On retourne en **1.**

La convergence de cet algorithme est assurée par la théorie développée par Conn, Gould et Toint dans [2], sous des hypothèses t.ès générales.
Il est à noter qu'une de ces hypothèses peut parfois ne pas être vérifiée : c'est la condition de complémentarité stricte, qui exige que si la variable x_i est à une de ses bornes à la solution, alors

$$(\nabla f)_i \neq 0 \tag{48}$$

Cette condition oblige la contrainte à être utile, en ce sens que si

$$(\nabla f)_i = 0$$

la présence ou non de la $i^{\text{ième}}$ contrainte n'a aucune influence sur la solution.
Nous verrons ce qui se passe lorsque cette condition n'est pas vérifiée dans les tests numériques.

4 Moindres carrés linéaires avec contraintes de borne

4.1 Enoncé

$$\begin{cases} \min_{x \in \mathbf{R}^n} \|Ax - b\|_2 \\ \text{s.c. } l_i \leq x_i \leq u_i \quad i = 1, \ldots, n \end{cases} \tag{49}$$

Par (6), on obtient un problème équivalent.

$$\begin{cases} \min_{x \in \mathbf{R}^n} f(x) = \frac{1}{2} x^t A^t A x - b^t A x \\ \text{s.c. } l_i \leq x_i \leq u_i \quad i = 1, \ldots, n \end{cases} \tag{50}$$

Remarque : pour avoir un problème non trivial, on suppose, sans perdre aucune généralité, que les contraintes sont compatibles (i.e. que $l_i \leq u_i \ \forall i$), et que le domaine admissible est non trivial (i.e. $\exists i$ tel que $l_i < u_i$)

4.2 Méthode de Björck

Åke Björck a décrit un algorithme pour résoudre le problème (49). (voir [6])
Cet algorithme utilise une méthode de contraintes actives : à chaque itération, on résoud le problème sans contrainte pour les variables libres[2].

[2]Variables qui ne sont pas à leur borne

Si la solution est admissible, on regarde si on est à l'optimum en calculant le gradient. Si on n'y est pas on libère une variable et on recommence.
Si la solution n'est pas admissible, on avance le long de la direction la reliant à l'itéré courant. On prendra comme nouvel itéré le dernier point admissible que l'on rencontre le long de cette direction.
On va utiliser une matrice de permutation Q qui va ordonner les colonnes de A de telle manière à ce que celles correspondant aux variables libres soient avant celles correspondant aux variables fixées.
Comme Q est une matrice de permutation, on a que $QQ^t = Q^tQ = I$.
On définit

$$Q^t x = \begin{pmatrix} x_{\mathcal{F}} \\ x_{\mathcal{B}} \end{pmatrix}$$

où $x_{\mathcal{F}}$ représente les variables libres ($\mathcal{F}$ pour *Free*), et $x_{\mathcal{B}}$ représente les variables fixées.
Si maintenant, on calcule la décomposition QR de la matrice permutée AQ, on obtient :

$$AQ = P \begin{pmatrix} R & S \\ 0 & U \\ 0 & 0 \end{pmatrix}$$

où, en posant $ff = \#\mathcal{F}$ et $bb = \#\mathcal{B} = n - ff$,

- $P \in \mathbf{R}^{m \times m}$ est orthogonale
- $R \in \mathbf{R}^{ff \times ff}$ est triangulaire supérieure
- $S \in \mathbf{R}^{ff \times bb}$
- $U \in \mathbf{R}^{bb \times bb}$ est triangulaire supérieure
- les 0 correspondent à des matrices nulles de dimensions appropriées

Si on définit

$$P^t b = \begin{pmatrix} c \\ d \\ e \end{pmatrix}$$

on peut aisément transformer le problème :

$$\begin{aligned}
\|Ax-b\| &= \|AQQ^tx-b\| \\
&= \left\| P\begin{pmatrix} R & S \\ 0 & U \\ 0 & 0 \end{pmatrix}\begin{pmatrix} x_{\mathcal{F}} \\ x_{\mathcal{B}} \end{pmatrix} - b \right\| \\
&= \left\| \begin{pmatrix} R & S \\ 0 & U \\ 0 & 0 \end{pmatrix}\begin{pmatrix} x_{\mathcal{F}} \\ x_{\mathcal{B}} \end{pmatrix} - P^tb \right\| \qquad (51) \\
&= \left\| \begin{pmatrix} R & S \\ 0 & U \\ 0 & 0 \end{pmatrix}\begin{pmatrix} x_{\mathcal{F}} \\ x_{\mathcal{B}} \end{pmatrix} - \begin{pmatrix} c \\ d \\ e \end{pmatrix} \right\| \\
&= \left\| \begin{matrix} Rx_{\mathcal{F}} + Sx_{\mathcal{B}} - c \\ Ux_{\mathcal{B}} - d \\ -e \end{matrix} \right\|
\end{aligned}$$

Ainsi, résoudre le problème sans contrainte pour les variables libres revient à résoudre le système triangulaire

$$Rx_{\mathcal{F}} = c - Sx_{\mathcal{B}}$$

Pour vérifier si on est à l'optimum, on doit calculer le gradient de la fonction

$$\tfrac{1}{2}\|AQQ^tx-b\|^2 = \tfrac{1}{2}x^tQQ^tA^tAQQ^tx - x^tQQ^tA^tb + \tfrac{1}{2}b^tb$$

Ce gradient vaut

$$\begin{aligned}
G &= QQ^tA^tAQQ^tx - QQ^tA^tb \\
&= Q\begin{pmatrix} R^t & 0 & 0 \\ S^t & U^t & 0 \end{pmatrix}P^tP\begin{pmatrix} R & S \\ 0 & U \\ 0 & 0 \end{pmatrix}\begin{pmatrix} x_{\mathcal{F}} \\ x_{\mathcal{B}} \end{pmatrix} - Q\begin{pmatrix} R^t & 0 & 0 \\ S^t & U^t & 0 \end{pmatrix}P^tb
\end{aligned}$$

$$Q^tG = \begin{pmatrix} R^t & 0 & 0 \\ S^t & U^t & 0 \end{pmatrix}\begin{pmatrix} R & S \\ 0 & U \\ 0 & 0 \end{pmatrix}\begin{pmatrix} x_{\mathcal{F}} \\ x_{\mathcal{B}} \end{pmatrix} - \begin{pmatrix} R^t & 0 & 0 \\ S^t & U^t & 0 \end{pmatrix}\begin{pmatrix} c \\ d \\ e \end{pmatrix}$$

$$\begin{pmatrix} G_{\mathcal{F}} \\ G_{\mathcal{B}} \end{pmatrix} = \begin{pmatrix} R^tR & R^tS \\ S^tR & S^tS + U^tU \end{pmatrix}\begin{pmatrix} x_{\mathcal{F}} \\ x_{\mathcal{B}} \end{pmatrix} - \begin{pmatrix} R^tc \\ S^tc + U^td \end{pmatrix}$$

Donc, en isolant la partie correpondant aux variables libres et celle correspondant aux variables

fixées :

$$\begin{aligned} G_{\mathcal{F}} &= R^t R x_{\mathcal{F}} + R^t S x_{\mathcal{B}} - R^t c \\ &= R^t (R x_{\mathcal{F}} + S x_{\mathcal{B}} - c) \\ G_{\mathcal{B}} &= S^t R x_{\mathcal{F}} + S^t S x_{\mathcal{B}} + U^t U x_{\mathcal{B}} - S^t c - U^t d \\ &= S^t (R x_{\mathcal{F}} + S x_{\mathcal{B}} - c) + U^t (U x_{\mathcal{B}} - d) \end{aligned} \tag{52}$$

Algorithme 7

Initialisations

On prend comme point de départ le centre du domaine admissible. Ainsi, aucune variable n'est à sa borne.

$$\begin{aligned} x &= \tfrac{l+u}{2} \\ \mathcal{F} &= \{1, 2, \ldots, n\} \\ \mathcal{B} &= \emptyset \\ Q &= I_n \end{aligned}$$

Remarque : c'est le choix fait par Björck pour le point de départ. On pourrait faire démarrer l'algorithme de n'importe quel point admissible (par exemple une approximation de la solution) à condition de bien définir les ensembles d'indices, et de permutter la matrice A au départ.

On calcule la décomposition QR de A en utilisant la librairie LINPACK (voir [8]) (On suppose que rang(A)=n) :

$$P^t A = \begin{pmatrix} R \\ 0 \end{pmatrix} \begin{matrix} n \\ m-n \end{matrix}$$

$$P^t b = \begin{pmatrix} c \\ e \end{pmatrix} \begin{matrix} n \\ m-n \end{matrix}$$

Itérations

On calcule la solution sans contrainte pour les variables libres

$$\begin{aligned} q &= R^{-1}(c - S x_{\mathcal{B}}) \\ q &\in \mathbf{R}^{//} \end{aligned}$$

On se replace dans $\mathbf{R}^n$

$$z_i = \begin{cases} q_i & \text{si } i \in \mathcal{F} \\ 0 & \text{sinon} \end{cases}$$

Il y a alors deux possibilités :

1. *La solution sans contrainte pour les variables libres est admissible*

$$l_i \leq z_i \leq u_i \quad \forall i \in \mathcal{F}$$

Dans ce cas, on vérifie si on est à la solution en calculant le gradient.
Pour la partie du gradient correspondant aux variables libres, pas besoin de la calculer.
En effet :

$$G_{\mathcal{F}} = R^t (R x_{\mathcal{F}} + S x_{\mathcal{B}} - c) = 0$$

vu qu'on a déterminé $x_{\mathcal{F}}$ pour l'annuler.
Ainsi, si toutes les variables sont libres, i.e. si $B = \emptyset$ on est à la solution
STOP

Si ce n'est pas le cas, on calcule le gradient pour les variables fixées :

$$\begin{aligned} G_B &= S^t(Rx_{\mathcal{F}} + Sx_B - c) + U^t(Ux_B - d) \\ &= U^t(Ux_B - d) \end{aligned}$$

On est à la solution si ,$\forall i \in B$,

$$-G_i \geq 0 \text{ et } x_i = u_i$$
$$\text{ou}$$
$$-G_i \leq 0 \text{ et } x_i = l_i$$

On calcule donc $\lambda = -G$ pour les variables fixées :

$$\lambda = U^t(d - Ux_B)$$

On définit γ tel que

$$\begin{aligned} \gamma_i &= 1 \text{ si } x_i = l_i \\ &= -1 \text{ si } x_i = u_i \end{aligned}$$

Remarque :
on ne doit pas recalculer les coéfficients γ à chaque itération. En effet, lors de la première itération toutes les variables sont libres, et donc les γ sont inutiles. Par après, ils seront mis à jour au fur et à mesure que l'on activera des variables.
Test : on est à l'optimum si

$$\gamma_i \lambda_i \leq 0 \ \forall i \in B$$

STOP

Si le test n'est pas vérifié, on n'est pas à l'optimum, il faut donc libérer une des variables pour lesquelles $\gamma_i \lambda_i > 0$.
On cherche l'indice t tel que

$$\gamma_t \lambda_t = \max_{i \in B} \gamma_i \lambda_i$$

On transfère l'indice t de l'ensemble B vers l'ensemble $\mathcal{F}$.
Comme Q est transformé, il faut remettre à jour la décomposition QR de AQ en utilisant LINPACK (voir [8])

$$P^t A Q = \begin{pmatrix} R & S \\ 0 & U \\ 0 & 0 \end{pmatrix}$$

et

$$P^t b = \begin{pmatrix} c \\ d \\ e \end{pmatrix}$$

Ensuite, on recommence.

2. *La solution sans contrainte pour les variables libres n'est pas admissible.*
   Dans ce cas, on suit la direction reliant l'itéré courant et cette solution sans contrainte, et on s'arrête dès qu'on atteint une borne.
   On calcule donc un pas α tel que

$$\alpha = \min_{i \in \mathcal{F}} \alpha_i$$

où

$$\alpha_i = \begin{cases} \frac{z_i - l_i}{z_i - x_i} & si\ z_i < l_i \\ \frac{u_i - z_i}{z_i - x_i} & si\ z_i > u_i \\ +\infty & sinon \end{cases}$$

On met x à jour :

$$x = x + \alpha(z - x)$$

On transfère de $\mathcal{F}$ vers $\mathcal{B}$ tous les indices q tels que $x_q = u_q$ ou $x_q = l_q$.
On met à jour les γ

$$\gamma_q = \begin{cases} 1 & si\ x_q = l_q \\ -1 & si\ x_q = u_q \end{cases}$$

Comme Q est transformé, il faut remettre à jour la décomposition QR de AQ en utilisant LINPACK (voir [8])

$$P^t A Q = \begin{pmatrix} R & S \\ 0 & U \\ 0 & 0 \end{pmatrix}$$

et

$$P^t b = \begin{pmatrix} c \\ d \\ e \end{pmatrix}$$

Et on recommence.

Remarques :

1. Les λ utilisés dans l'algorithme pour le test d'optimalité sont en fait les multiplicateurs de Lagrange.

2. Cette méthode ne tient absolument pas compte d'une éventuelle structure creuse de la matrice A. On pourrait envisager l'emploi d'une décomposition QR creuse. Ainsi, la comparaison entre cette méthode et le nouvel algorithme que nous allons décrire à la section 4.4 serait plus équitable.

3. Le plus gros inconvénient de cette méthode est qu'elle ne peut désactiver qu'une seule contrainte à la fois. De ce fait, si l'utilisateur désire commencer les itérations d'un point situé sur la frontière du dpmaine admissible, l'algorithme prendra beaucoup d'itérations

pour désactiver les contraintes inutiles.
De plus, même si la méthode peut théoriquement activer autant de contraintes qu'elle l'estime nécessaire, on remarque (voir les résultats numériques à la section 5) qu'en général, il n'y en a qu'une qui est activée par itération.
Ainsi, plus il y a de contraintes actives à la solution, plus le nombre d'itérations sera élevé.

4.3 Méthode de Lötstedt

Voici l'algorithme proposé par Per Lötstdedt dans [7], dont Björck s'est inspiré pour développer le sien.
On désire résoudre le problème suivant :

$$\begin{aligned} &\min \|Ax+b\|_2 \\ &\text{s.c.} \quad c_i \le x_i \le d_i \quad i=1\ldots k \\ &\qquad x_i \ge c_i \qquad\quad i=k+1\ldots l \end{aligned} \tag{53}$$

avec

$$\begin{aligned} &A \in \mathbf{R}^{m\times n} \\ &b \in \mathbf{R}^m \\ &x \in \mathbf{R}^n \\ &l \le n \end{aligned}$$

De plus, lorsqu'il a trouvé l'ensemble des solutions $\mathcal{M}$, Lötstedt s'attaque encore au problème de la solution minimale, i.e. :

$$\min_{x\in\mathcal{M}} \|x\|_2$$

mais nous ne nous attarderons pas sur ce sujet.
Voici quelques définitions utilisées dans l'algorithme :

- Soit x admissible
- Soient les ensembles d'indices $\mathcal{F}, \mathcal{P}$ et $\mathcal{X}$:
  $\mathcal{X}$: indices des variables fixées

$$\mathcal{X} = \{i \in \{1\ldots n\} \mid x_j = c_j \text{ ou } x_j = d_j\}$$

  $\mathcal{F}$: indices des variables libres

$$\mathcal{F} = \{i \mid c_i < x_i < d_i\}$$

  $\mathcal{P}$: indices des variables passives

$$\mathcal{P} = \{1\ldots n\}/\{\mathcal{X} \cup \mathcal{F}\}$$

- Soit $n_{\mathcal{F}}$ le nombre de variables libres.
- Soit $E_{\mathcal{F}} \in \mathbf{R}^{n\times n_{\mathcal{F}}}$ tel que le vecteur $x_{\mathcal{F}} = E_{\mathcal{F}}^t x$ contienne toutes les variables libres, et $A_{\mathcal{F}} = AE_{\mathcal{F}}$ contienne les colonnes a_j de A, avec $j \in \mathcal{F}$.

On va minimiser

$$\|A_{\mathcal{F}} x_{\mathcal{F}} + b'\|$$

avec

$$b' = b + \sum_{j \in \overline{X \cup P}} a_j x_j$$

en gardant x admissible.

Algorithme 8
Initialisations

1. *On choisit x admissible*
2. *On pose $\mathcal{X} = \emptyset$*
3. *On choisit le premier $\mathcal{F}$ et $A_{\mathcal{F}}$*
4. *On calcule le résidu $\|Ax + b\|$*

Itérations

1. *Si $\mathcal{F} \neq \emptyset$, on calcule une nouvelle direction de descente p en résolvant*

$$A^t_{\mathcal{F}} A_{\mathcal{F}} p + A^t_{\mathcal{F}} r = 0 \tag{54}$$

   Sinon, aller en 5.

2. *Soit $\bar{\theta} p$ ($\bar{\theta} \geq 0$) le plus grand pas que l'on puisse faire le long de p dans l'espace des variables libres sans violer de contraintes.*
   On prend $\theta = \min(\bar{\theta}, 1)$.
   θ satisfait :

$$\begin{aligned} r_i^- &= \tfrac{c_i - x_i}{p_i} \quad i \in \mathcal{F}_- = \{j \ tq \ j \leq l, j \in \mathcal{F}, p_j < 0\} \\ r_i^+ &= \tfrac{d_i - x_i}{p_i} \quad i \in \mathcal{F}_+ = \{j \ tq \ j \leq k, j \in \mathcal{F}, p_j > 0\} \end{aligned}$$

$$\theta = \min(\min_{i \in \mathcal{F}_-} r_i^-, \min_{i \in \mathcal{F}_+} r_i^+, 1) \tag{55}$$

3. *Mise à jour de x et de r*

$$\begin{aligned} x_i &= x_i + \theta p_i \quad i \in \mathcal{F} \\ r &= r + \theta A_{\mathcal{F}} p \end{aligned}$$

4. *Si $r_i^- > 1$, $i \in \mathcal{F}_-$ et $r_i^+ > 1$, $i \in \mathcal{F}_+$ au pas 2, alors aller en 5.*
   Pour chaque x_j atteignant une de ses bornes au pas 3, transférer j de $\mathcal{F}$ dans $\mathcal{X}$ et retirer la colonne correspondante de $A_{\mathcal{F}}$.
   Retourner en 1.

5. *Calculer le gradient Δ de la fonction objectif :*

$$\Delta = A^t r = A^t A x + A^t b$$

6. Soit

$$\delta_i = \begin{cases} \Delta_i & si\ x_i = d_i \\ -\Delta_i & si\ x_i = c_i \end{cases} \quad i \in \mathcal{X}$$

Trouver γ tel que

$$\gamma = \max(\max_{i \in \mathcal{X}} \delta_i, \max_{i \in P} |\Delta_i|)$$

et j tel que

$$\gamma = \delta_j, j \in \mathcal{X}$$
ou
$$\gamma = |\Delta_j|, j \in P$$

- *Si $\gamma > \varepsilon$ (où ε est la tolérance sur l'erreur), alors on peut diminuer la fonction objectif si j devient une variable libre.*
  On transfère j de $\mathcal{X}$ ou P dans $\mathcal{F}$, et on ajoute la colonne correspondante dans $A_{\mathcal{F}}$.
  Retourner en 1
- *Si $\gamma \leq \varepsilon$, on a une bonne approximation de l'optimum*

7. Fin de l'algorithme

Le système d'équations linéaires (54) peut être résolu :

- soit par une méthode directe basée sur les transformations de Householder (comme dans la méthode de Björck, section 4.2)
- soit par une méthode de gradients conjugués préconditionnés

Lorsqu'on applique une méthode de gradients conjugués, on introduit une matrice de préconditionnement $C_{\mathcal{F}}$, et on applique les gradients conjugués à

$$C_{\mathcal{F}}^{-t} A_{\mathcal{F}}^t A_{\mathcal{F}} C_{\mathcal{F}}^{-1} v + C_{\mathcal{F}}^{-t} A_{\mathcal{F}}^t r = 0$$

Si on prend

$$p = C_{\mathcal{F}}^{-1} v$$

on a une solution de (54)
Lötstedt utilise comme critère d'arrêt pour sa routine de gradients conjugués le test :

$$\|A_{\mathcal{F}}^t A_{\mathcal{F}} p_* + A_{\mathcal{F}}^t r\| < \frac{\varepsilon}{\|C_{\mathcal{F}}^{-1}\|}$$

Notons qu'il s'attache surtout au cas où l'on doit résoudre une suite de problèmes où les données varient peu d'un problème à l'autre. Dans ce cas, la même matrice de conditionnement peut servir pour plusieurs problèmes consécutifs.
Il propose comme matrice de préconditionnement la matrice triangulaire supérieure résultat de la décomposition QR de $A_{\mathcal{F}}$.
Il est évident que cette proposition n'est valable que dans le cas bien spécifique d'une suite de problèmes, et qu'elle n'est pas du tout applicable dans le cas d'un problème particulier. En effet, le calcul de la décomposition QR peut nous permettre directement de résoudre l'équation (54) sans passer par les gradients conjugués.

4.4 Nouvel algorithme proposé

On va reprendre la démarche proposée à la section 3.5 pour minimiser une fonction non linéaire quelconque, en l'adaptant au problème quadratique.

4.4.1 Adaptations

1. Comme il a déjà été précisé, la fonction étant égale partout au modèle, on n'utilise pas de région de confiance.

2. Le gradient de la fonction (7) au point x est

$$g(x) = A^t Ax - A^t b$$

   Remarque : on retrouve ici l'expression des équations normales (2). Donc, résoudre les équations normales revient à annuler le gradient de la fonction objectif. On est certain que le point qui annule le gradient est le minimum de la fonction, car la matrice hessienne $(A^t A)$ étant définie positive, la fonction (7) est convexe.

3. On ne va pas calculer d'approximation du hessien, car on en possède l'expression analytique :

$$H(x) = A^t A \quad \forall\, x \in \mathbb{R}^n$$

4. L'algorithme des gradients conjugués est remplacé par l'algorithme LSQR de Paige et Saunders décrit à la section 2.5.
   Le théorème 6 montre qu'en fait, LSQR génére théoriquement les mêmes itérés que l'algorithme des gradients conjugués. La motivation du choix de LSQR est dûe à la structure creuse de la matrice A, et au peu d'opérations qu'il demande (3 produits matrice-vecteur par itération).
   LSQR étant une méthode prévue pour les cas sans contrainte, elle pourrait générer des points non admissibles. Si c'est le cas, on arrête l'algorithme, et on projette le premier itéré non admissible sur la région admissible :
   Soit x un point non admissible généré par LSQR.
   Soit $d = x - x_k^{\mathcal{F}}$.
   On calcule le plus long pas λ que l'on puisse faire à partir de $x_k^{\mathcal{F}}$, le long de d, sans sortir du domaine admissible.
   On prendra comme "résultat" du LSQR le point $x_k^{\mathcal{F}} + \lambda d$. Comme LSQR est une méthode de descente (voir le théorème 5)

4.4.2 Remarques

1. **Choix du point de départ**
   Pour épargner une évaluation du produit $A^t Ax$ intervenant dans le calcul du gradient, on aimerait pouvoir toujours prendre $x_0 = 0_{\mathbb{R}^n}$ comme point de départ.
   Si $0_{\mathbb{R}^n} \in \mathcal{F}$ on prendrait $-A^t b$ comme gradient initial, sinon, on translaterait le problème de telle manière à amener $0_{\mathbb{R}^n}$ dans le domaine admissible $\mathcal{F}$.

Nous allons montrer que l'on ne gagne rien à translater le problème.
Problème initial (50) :

$$\begin{cases} \min_{x \in \mathbf{R}^n} f(x) = \frac{1}{2} x^t A^t A x - b^t A x \\ \text{s.c. } l_i \leq x_i \leq u_i \quad i = 1, \ldots, n \end{cases}$$

Supposons $0_{\mathbf{R}^n}$ non admissible, i.e. $0_{\mathbf{R}^n} \notin \mathcal{F}$
Comme l'ensemble admissible est supposé non vide, on peut trouver un vecteur x_0, non nul, admissible.
Posons $y = x - x_0$, i.e. $x = y + x_0$.
Le problème devient :

$$\begin{cases} min_{y \in \mathbf{R}^n} f(y) = \frac{1}{2} y^t A^t A y + x_0^t A^t A y - b^t A y \ + \text{constantes} \\ \text{s.c. } l_i - x_{0i} \leq y \leq u_i - x_{0i} \quad i = 1, \ldots, n \end{cases}$$

On définit $\tilde{\mathcal{F}}$ comme étant la région admissible de ce nouveau problème.

$$y \in \tilde{\mathcal{F}}$$
$$\Longleftrightarrow$$
$$l_i - x_{0i} \leq y \leq u_i - x_{0i} \quad i = 1, \ldots, n$$

Il est clair que $0 \in \tilde{\mathcal{F}}$. Malheureusement, le gradient en ce point n'est plus $-A^t b$ mais $A^t A x_0 - A^t b$ qui demande aussi une évaluation de $A^t A$.
Il est donc inutilement coûteux de translater ainsi le problème.

2. Mises à jour des dérivées directionnelles

Dans l'algorithme de Conn, Gould et Toint 3.5, on utilise un vecteur

$$g = g_k - B_k x_k$$

dans la mise à jour des dérivées directionnelles pour le calcul du point de Cauchy généralisé.
Voyons ce que ce vecteur devient dans notre problème spécifique :

$$\begin{aligned} g_k &= A^t A x_k - A^t b \\ B_k &= A^t A \end{aligned}$$

et donc

$$\begin{aligned} g &= A^t A x_k - A^t b - A^t A x_k \\ &= -A^t b \end{aligned}$$

Lors de l'implémentation, le vecteur $g = -A^t b$ sera calculé une fois pour toutes avant de commencer les itérations.

3. Stockage de la matrice A

Si on veut traiter des problèmes où la matrice A est grande et creuse, il est avantageux d'exploiter cette structure creuse lors du stockage. En effet, il est inutile de stocker un grand nombre d'éléments nuls.
La matrice A va être stockée grâce à trois tableaux (unidimensionnels) : un tableau double précision, et deux tableaux d'entiers.

- Le tableau double précision (appelé A) va contenir les éléments non nuls de la matrice : d'abord ceux de la première colonne, ensuite ceux de la deuxième, etc...
  La matrice est donc stockée colonne par colonne.
  C'est un choix que l'on a fait : on pourrait très bien utiliser un stockage ligne par ligne.
- Le premier tableau d'entiers (appelé INDLIGNE) aura la même longueur que le tableau A.
  INDLIGNE(i) contient le numéro de la ligne où se trouve l'élément A(i) dans la matrice *A*.
- Le second tableau d'entiers (appelé DEBCOL), de longueur n+1 (où n est le nombre de colonnes de la matrice), est un tableau de pointeurs.
  En effet, DEBCOL(i) (pour i=1,...,n) contiendra l'indice du premier élément de la colonne i dans le tableau A.
  DEBCOL(n+1) contiendra le nombre d'éléments non nuls de la matrice, plus un.
  Ainsi, on peut aisément connaître le nombre d'éléments non nuls dans une colonne i quelconque. Ce sera :

$$DEBCOL(i+1) - DEBCOL(i)$$

Exemple :

$$A = \begin{pmatrix} 1.2 & 0 & 0 & 0 \\ 0 & 0 & 4.5 & 0 \\ 0 & 3.4 & 5.6 & 0 \\ 0 & 0 & 6.7 & 0 \\ 2.3 & 0 & 0 & 7.8 \end{pmatrix}$$

A = (1.2 , 2.3 , 3.4 , 4.5 , 5.6 , 6.7 , 7.8)
INDLIGNE = (1 , 5 , 3 , 2 , 3 , 4 , 5)

DEBCOL = (1,3,4,7,8)

Bien entendu, cette manière de stocker les matrices n'est pas habituelle. Il a donc fallu écrire une procédure effectuant un produit matrice-vecteur, qui tienne compte du stockage spécial de la matrice.

4. Point de départ du LSQR

L'algorithme LSQR est prévu pour avoir le point 0_{R^p} comme point de départ (où p est la dimension du sous-espace des variables libres).
Si ce vecteur nul n'est pas admissible, on va devoir translater le problème de manière à le rendre admissible.
Plutôt que de faire un test à chaque itération, le problème va être systématiquement trans laté.
Cette translation va envoyer le point de Cauchy généralisé (réduit au sous-espace des variables libres) sur 0_{R^p}.
Il y a trois avantages à cela :

(a) On est certain que le nouveau point issu de l'algorithme LSQR va provoquer une diminution de la fonction objectif par rapport au point de Cauchy généralisé, ce qui est exigé par la théorie de convergence (voir [2]).

(b) Le point de Cauchy généralisé est toujours admissible, ce qui assure que le vecteur nul sera admissible pour le problème translaté.

(c) Le PCG constitue une information sur la région où se trouve le minimum. En effet, c'est le premier minimum local de la fonction le long de la projection de la plus forte pente sur la région admissible.
Il semble plus logique de partir de ce point qui réduit déjà la fonction objectif, plutôt que de choisir un point admissible quelconque.

En terme de coûts, la translation du problème demande un produit matrice-vecteur. En effet, soit le problème initial

$$\min_{x \in \mathbf{R}^n} \|Ax - b\|$$
$$\text{s.c. } l \leq x \leq u$$

On translate le problème :

$$y = x - x_k^i$$

$$\min_{y \in \mathbf{R}^n} \|A(y + x_k^i) - b\|$$
$$\text{s.c. } l - x_k^i \leq y \leq u - x_k^i$$

$$\min_{y \in \mathbf{R}^n} \|Ay - (b - Ax_k^i)\|$$
$$\text{s.c. } l - x_k^i \leq y \leq u - x_k^i$$

Le nouveau second membre du problème est donc :

$$b_{tr} = b - Ax_k^i$$

4.4.3 Algorithme

Voici l'algorithme proposé pour résoudre le problème des moindres carrés linéaires avec contraintes de bornes (49)

Algorithme 9

1. **Initialisations**

(a) Si $0_{\mathbf{R}^n}$ *est admissible alors*
$x_0 = 0_{\mathbf{R}^n}$
Sinon
$x_0 = l$
Remarque : on peut prendre n'importe quel point admissible

(b) Si $0_{\mathbb{R}^n}$ est admissible alors

$g_0 = -A^t b$

Sinon

$g_0 = A^t A x_o - A^t b$

2. Itérations

Pour $k = 0, \ldots$

(a) Calcul du gradient projeté

$\overline{g}_k = P(x_k - g_k) - x_k$

Si $\|\overline{g}_k\|_2 \leq \epsilon$ STOP

(b) Calcul du point de Cauchy généralisé

i. Initialisations

On pose

$$x = x_k$$

$$d_i = \begin{cases} 0 & \text{si } (x_k)_i \text{ est à sa borne inférieure et } [-(g_k)_i] < 0 \\ 0 & \text{si } (x_k)_i \text{ est à sa borne supérieure et } [-(g_k)_i] > 0 \\ -(g_k)_i & \text{sinon} \end{cases}$$

$$f' = g_k^t d$$

$$f'' = d^t A^t A d = \|Ad\|^2$$

Si $f' \geq 0$ aller en 2(b)v (On a trouvé le PCG)

ii. Recherche du point de cassure suivant

On cherche le plus long pas que l'on puisse faire le long de d sans sortir de la région admissible : λ

Soit I l'ensemble des indices des variables qui rencontrent leurs bornes en $x + \lambda d$.

iii. Le point de Cauchy est-il sur cet arc ?

Si $f'' > 0$ et $0 < -\frac{f'}{f''} < \lambda$ alors

$x = x - \frac{f'}{f''} d$

aller en 2(b)v

iv. Mise à jour des dérivées directionnelles

$x = x + \lambda d$

$z = A^t \sum_{i \in I} (d_i a_i)$ où a_i est la $i^{ième}$ colonne de A

$f' = f' + \lambda f'' - z^t x + \sum_{i \in I} d_i (A^t b)_i$

$f'' = f'' + \sum_{i \in I} d_i z_i - 2 z^t d$

$d_i = 0 \ \forall i \in I$

Si $f^t \geq 0$
aller en 2(b)v
Sinon
retourner en 2(b)ii

v. **On a trouvé le PCG**
$x_k^t = x$

(c) **Calcul du nouvel itéré**

i. Soit $J = \{1, \ldots, n\} - I(x_k^t)$ *où* $I(.)$ *est défini en (44)*

ii. On translate le problème pour faire démarrer les itérations de LSQR à partir du point de Cauchy

$$b_{tr} = b - Ax_k^t$$
$$u_{tr} = u - x_k^t$$
$$l_{tr} = l - x_k^t$$

iii. On applique LSQR au problème translaté, réduit aux variables indexées par J *:*

$$\min_{(x_i)_{i \in J}} \|Ax - b_{tr}\|$$

On arrête l'algorithme LSQR si :

- *Un itéré viole les bornes* $[l_{tr}, u_{tr}]$
- *On obtient une solution du problème réduit*
- *On dépasse un certain nombre d'itérations*
- *LSQR ne peut pas trouver de solution parce que soit l'estimation du conditionnement de la matrice, soit la norme de l'itéré est trop élevée. Dans ce cas, on ne poursuivra pas l'algorithme principal.*

Soit x_{LSQR} *le point donné par l'algorithme et* x^* *tel que*

$$(x^*)_i = \begin{cases} (x_{LSQR})_i & si\ i \in J \\ (x_k^t)_i & si\ i \notin J \end{cases}$$

Si x^* *n'est pas admissible alors*
on calcule le plus grand λ *tel que* $x_k^t + \lambda(x^* - x_k^t)$ *soit admissible*
on pose $x_k = x_k^t + \lambda(x^* - x_k^t)$
Sinon
$x_k = x^*$

(d) **Calcul du gradient**

$$g_k = A^t A x_k - A^t b$$

Remarque :
Un grand avantage de cette méthode est qu'à chaque itération, elle peut activer ou désactiver autant de contraintes que nécessaire. Ce qui fait qu'elle détecte l'ensemble des indices actifs (i.e.

les indices correspondant aux contraintes actives) à la solution en très peu d'itérations (voir les résultats obtenus à la section 5)

5 Résultats numériques

5.1 Problèmes testés

Pour définir un problème on doit avoir :

1. **Une matrice**
   On a utilisé trois sortes de matrices :

   (a) Des matrices aléatoires, i.e. des matrices dont la structure et les éléments sont déterminés aléatoirement[3]. On impose

   - La dimension de la matrice
   - Le nombre d'éléments non nuls par colonne

   Exemple : Matrice 10 x 5 avec 3 éléments non nuls par colonne :

$$\begin{pmatrix} 0 & * & 0 & 0 & 0 \\ * & 0 & * & * & 0 \\ 0 & 0 & 0 & 0 & * \\ 0 & * & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 \\ 0 & 0 & 0 & * & * \\ * & 0 & 0 & * & 0 \\ 0 & 0 & * & 0 & 0 \\ 0 & 0 & 0 & 0 & * \\ * & 0 & * & 0 & 0 \end{pmatrix}$$

   On a testé quatre matrices de cette forme :

   i. Matrice 100 x 50 (10 éléments non nuls par colonne)
   ii. Matrice 500 x 100 (20 éléments non nuls par colonne)
   iii. Matrice 1000 x 400 (30 éléments non nuls par colonne)
   iv. Matrice 1000 x 800 (10 éléments non nuls par colonne)

   (b) Des matrices bandes, dont les éléments sont déterminés aléatoirement, mais où la structure est définie en fonction de la largeur de la bande.

[3] Les éléments sont choisis entre -100 et 100

Exemple : Matrice 10 x 5 avec une largeur de bande de 2 :

$$\begin{pmatrix} * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \\ 0 & * & * & * & 0 \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

On a testé quatre matrices de cette forme :

i. Matrice 100 x 50 (largeur de bande : 5)

ii. Matrice 500 x 100 (largeur de bande : 10)

iii. Matrice 1000 x 400 (largeur de bande : 15)

iv. Matrice 1000 x 800 (largeur de bande : 5)

(c) Des matrices pleines, i.e. des matrices où il n'y a aucun élément nul.
On a testé quatre matrices de cette forme :

i. Matrice 100 x 50

ii. Matrice 1000 x 5

iii. Matrice 80 x 80

iv. Matrice 176 x 176

2. Un terme indépendant

Dans tous les cas, le terme indépendant b est déterminé aléatoirement.

3. Des bornes

Les bornes seront les mêmes pour toutes les variables. Pour chaque matrice, on a imposé quatre contraintes différentes :

(a) $-10^5 \leq x \leq 10^{-5}$

Etant donné que les élements des matrices et des termes indépendants sont entre -100 et 100, ces bornes ne sont pas représentatives. Elles permettront de tester l'algorithme dans des cas sans contraintes. De plus, lorsqu'une variable a tendance à "exploser" lorsque le problème est mal conditionné, ces bornes l'empêchent de prendre des valeurs trop grandes (en valeur absolue).

(b) $-10^5 \leq x \leq 0$

Ce sont essentiellement des contraintes de non-positivité.

(c) $-1 \leq x \leq 1$

Ce sont de véritables contraintes de bornes. Mais il s'avérera lors des tests que peu de contraintes seront actives à la solution. C'est pourquoi on va encore réduire le domaine.

(d) $0 \leq x \leq 1$

Ces bornes définissent un domaine assez petit pour que l'activité des contraintes joue un rôle lors du déroulement de l'algorithme.

Remarque : pour tous les tests concernant la nouvelle méthode , nous avons demandé une précision de 10^{-8}, i.e. que l'algorithme s'arrête lorsque la norme du gradient projeté est plus petite que 10^{-8}. Quant aux tests d'arrêt du LSQR, nous avons donné comme valeur à ATOL et BTOL 10^{-17}. Cette valeur peut paraître excessive, mais il s'est avéré que la norme du gradient à la solution était de l'ordre de 10^{-9} avec ces valeurs. De plus petites valeurs de ATOL et BTOL diminuent sensiblement la précision sur la norme du gradient. Comme le test d'optimalité du programme principal est basé sur la norme du gradient, nous avons estimé qu'il fallait le même ordre de précision pour les itérations majeures et mineures.

5.2 Algorithmes utilisés

Nous avons testé les différents problèmes définis ci-dessus avec deux algorithmes différents :

1. L'algorithme de Björck, décrit à la section 4.2
2. Le nouvel algorithme proposé à la section 4.4, dans lequel la structure creuse de la matrice est exploitée.

Il faudra bien faire attention lors des comparaisons entre les résultats : l'algorithme de Björck résoud les sous-probèmes par une méthode directe (décomposition QR) , sans tenir compte du creux, alors que le nouvel algorithme utilise une méthode itérative de type gradients conjugués (LSQR).
Ce qui nous intéresse essentiellement est le comportement de la nouvelle méthode pour trouver l'ensemble actif à la solution.
A la différence de l'algorithme de Björck qui active ou désactive une contrainte à la fois, l'algorithme basé sur le point de Cauchy généralisé peut activer ou désactiver autant de contraintes que nécessaire.
Il sera donc intéressant de comparer le nombre d'itérations majeures effectué par chacune des méthodes.
Les comparaisons entre les temps d'exécution (temps CPU) devront se faire avec prudence et ne devront pas mener à des conclusions trop hâtives.
Remarques :
Dans certains cas, le programme utilisant la méthode de Björck a dû s'arrêter pour cause d'overflow. Nous l'avons signalé dans les tableaux concernés.

5.3 Résultats

Les tableaux 1 à 12 donnent pour chacune des deux méthodes :

- Le nombre d'itérations majeures (It. maj.)
- Le nombre de variables actives à la solution (Var. act.)
- Le temps CPU total donné en secondes (CPU). Il y a une certaine réserve à faire quant à cette donnée : le temps CPU dépend évidemment de l'implémentation de la méthode, mais aussi de la charge de la machine lorsque le programme a tourné.
- Le temps CPU utilisé à la résolution des sous-problèmes, donné en secondes également (CPU min). Pour la nouvelle méthode il s'agit du temps utilisé par l'algorithme LSQR, alors que pour la méthode de Björck, il s'agit du temps utilisé par la décomposition QR et ses mises à jour successives (à l'exclusion de la résolution du système triangulaire).

Pour la nouvelle méthode, on donne également le nombre total d'itérations mineures, i.e. les itérations de LSQR. (It. min.)

- Les tableaux 1 à 4 donnent les résultats pour les matrices aléatoires (pages 45 et suivantes)
- Les tableaux 5 à 8 donnent les résultats pour les matrices bandes (pages 49 et suivantes)
- Les tableaux 9 à 12 donnent les résultats pour les matrices pleines (pages 53 et suivantes)

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	56
	Var. act.	0	0
	CPU	0.68	0.60
	CPU min	0.67	0.56
$[-10^5, 0]$	It. Maj.	26	4
	It. Min.	-	32
	Var. act.	25	25
	CPU	1.14	0.33
	CPU min	0.92	0.25
$[-1, 1]$	It. Maj.	3	4
	It. Min.	-	68
	Var. act.	2	2
	CPU	0.71	0.80
	CPU min	0.67	0.70
$[0, 1]$	It. Maj.	22	5
	It. Min.	-	69
	Var. act.	19	19
	CPU	1.06	0.66
	CPU min	0.83	0.52

Table 1: Matrice aléatoire 100 x 50

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	43
	Var. act.	0	0
	CPU	13.77	1.54
	CPU min	13.74	1.42
$[-10^5, 0]$	It. Maj.	55	4
	It. Min.	-	30
	Var. act.	54	54
	CPU	16.81	0.96
	CPU min	15.61	0.65
$[-1, 1]$	It. Maj.	1	1
	It. Min.	-	43
	Var. act.	0	0
	CPU	13.77	1.51
	CPU min	13.73	1.41
$[0, 1]$	It. Maj.	46	2
	It. Min.	-	30
	Var. act.	45	45
	CPU	16.43	1.39
	CPU min	15.38	0.71

Table 2: Matrice aléatoire 500 x 100

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	76
	Var. act.	0	0
	CPU	445.56	13.50
	CPU min	445.22	13.11
$[-10^5, 0]$	It. Maj.	199	7
	It. Min.	-	45
	Var. act.	196	196
	CPU	605.21	9.72
	CPU min	548.33	5.01
$[-1, 1]$	It. Maj.	1	1
	It. Min.	-	76
	Var. act.	0	0
	CPU	452.05	13.89
	CPU min	451.72	13.50
$[0, 1]$	It. Maj.	212	7
	It. Min.	-	79
	Var. act.	209	209
	CPU	615.31	55.67
	CPU min	554.93	8.06

Table 3: Matrice aléatoire 1000 x 400

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	309
	Var. act.	0	0
	CPU	1495.28	48.41
	CPU min	1493.45	48.07
$[-10^5, 0]$	It. Maj.	416	17
	It. Min.	-	89
	Var. act.	395	395
	CPU	2735.93	27.77
	CPU min	2252.80	10.02
$[-1, 1]$	It. Maj.	81	32
	It. Min.	-	231
	Var. act.	78	78
	CPU	1745.78	56.20
	CPU min	1642.26	35.77
$[0, 1]$	It. Maj.		16
	It. Min.		87
	Var. act.	Overflow	408
	CPU		415.82
	CPU min		9.68

Table 4: Matrice aléatoire 1000 x 800

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	83
	Var. act.	0	0
	CPU	0.38	0.79
	CPU min	0.36	0.75
$[-10^5, 0]$	It. Maj.	33	4
	It. Min.	-	28
	Var. act.	28	28
	CPU	1.00	0.29
	CPU min	0.71	0.19
$[-1, 1]$	It. Maj.	6	12
	It. Min.	-	152
	Var. act.	5	5
	CPU	0.48	1.57
	CPU min	0.41	1.35
$[0, 1]$	It. Maj.	33	4
	It. Min.	-	58
	Var. act.	24	24
	CPU	0.94	0.51
	CPU min	0.69	0.38

Table 5: Matrice bande 100 x 50

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	83
	Var. act.	0	0
	CPU	7.69	5.51
	CPU min	7.65	5.40
$[-10^5, 0]$	It. Maj.	61	12
	It. Min.	-	62
	Var. act.	53	53
	CPU	10.39	2.28
	CPU min	9.12	1.38
$[-1, 1]$	It. Maj.	17	17
	It. Min.	-	188
	Var. act.	10	10
	CPU	8.60	6.81
	CPU min	8.25	5.74
$[0, 1]$	It. Maj.	60	12
	It. Min.	-	63
	Var. act.	55	55
	CPU	10.97	4.93
	CPU min	9.77	1.36

Table 6: Matrice bande 500 x 100

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	753
	Var. act.	0	0
	CPU	185.79	118.72
	CPU min	185.49	118.35
$[-10^5, 0]$	It. Maj.	242	15
	It. Min.	-	156
	Var. act.	208	208
	CPU	350.64	24.95
	CPU min	284.09	14.96
$[-1, 1]$	It. Maj.	63	39
	It. Min.	-	578
	Var. act.	27	27
	CPU	229.17	100.17
	CPU min	211.08	89.25
$[0, 1]$	It. Maj.	259	17
	It. Min.	-	92
	Var. act.	194	194
	CPU	372.57	117.11
	CPU min	299.03	9.64

Table 7: Matrice bande 1000 x 400

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	2169
	Var. act.	0	0
	CPU	600.79	304.13
	CPU min	599.15	303.80
$[-10^5, 0]$	It. Maj.		22
	It. Min.		160
	Var. act.	Overflow	378
	CPU		33.44
	CPU min		16.73
$[-1, 1]$	It. Maj.	210	39
	It. Min.	-	578
	Var. act.	125	27
	CPU	1116.40	138.91
	CPU min	895.87	84.16
$[0, 1]$	It. Maj.		16
	It. Min.		112
	Var. act.	Overflow	432
	CPU		426.67
	CPU min		11.06

Table 8: Matrice bande 1000 x 800

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	56
	Var. act.	0	0
	CPU	0.78	3.11
	CPU min	0.76	2.97
$[-10^5, 0]$	It. Maj.	17	5
	It. Min.	-	97
	Var. act.	16	16
	CPU	1.07	4.16
	CPU min	0.88	3.54
$[-1, 1]$	It. Maj.	1	1
	It. Min.	-	56
	Var. act.	0	0
	CPU	0.74	3.13
	CPU min	0.72	2.98
$[0, 1]$	It. Maj.	27	2
	It. Min.	-	26
	Var. act.	26	26
	CPU	1.05	1.07
	CPU min	0.88	0.74

Table 9: Matrice pleine 100 x 50

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	5
	Var. act.	0	0
	CPU	0.15	0.52
	CPU min	0.13	0.35
$[-10^5, 0]$	It. Maj.	2	1
	It. Min.	-	4
	Var. act.	1	1
	CPU	0.16	0.38
	CPU min	0.15	0.23
$[-1, 1]$	It. Maj.	1	1
	It. Min.	-	5
	Var. act.	0	0
	CPU	0.15	0.52
	CPU min	0.15	0.35
$[0, 1]$	It. Maj.	5	1
	It. Min.	-	1
	Var. act.	4	4
	CPU	0.15	0.23
	CPU min	0.15	0.04

Table 10: Matrice pleine 1000 x 5

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	143
	Var. act.	0	0
	CPU	1.23	10.01
	CPU min	1.20	9.85
$[-10^5, 0]$	It. Maj.	51	7
	It. Min.	-	50
	Var. act.	40	40
	CPU	3.01	3.10
	CPU min	2.15	1.95
$[-1, 1]$	It. Maj.	12	8
	It. Min.	-	376
	Var. act.	3	3
	CPU	1.56	26.33
	CPU min	1.32	25.23
$[0, 1]$	It. Maj.	45	9
	It. Min.	-	59
	Var. act.	38	38
	CPU	2.97	4.01
	CPU min	2.15	2.46

Table 11: Matrice pleine 80 x 80

Bornes		Méthode de Björck	Nouvelle méthode
$[-10^5, 10^5]$	It. Maj.	1	1
	It. Min.	-	300
	Var. act.	0	0
	CPU	11.63	97.71
	CPU min	11.55	96.88
$[-10^5, 0]$	It. Maj.	98	12
	It. Min.	-	83
	Var. act.	83	83
	CPU	27.06	30.68
	CPU min	20.84	15.83
$[-1, 1]$	It. Maj.	9	4
	It. Min.	-	529
	Var. act.	2	2
	CPU	12.96	173.55
	CPU min	12.35	170.70
$[0, 1]$	It. Maj.	109	13
	It. Min.	-	79
	Var. act.	90	90
	CPU	27.41	28.22
	CPU min	20.79	14.04

Table 12: Matrice pleine 176 x 176

On peut constater plusieurs choses à partir de ces résultats :

- La nouvelle méthode demande peu d'itérations majeures par rapport au nombre de variables actives à la solution, et même très peu lorsque ce nombre est élevé. Par exemple, pour la matrice aléatoire 1000 x 800 avec les contraintes [0,1], il lui a fallu 16 itérations majeures pour détecter les 408 contraintes qui sont actives à la solution.

- Par contre, pour la méthode de Björck, le nombre d'itérations est de l'ordre du nombre de variables actives à la solution, pour les petits problèmes. Par exemple pour la matrice pleine 100 x 50 avec les contraintes [0,1], il faut 27 itérations pour trouver 26 contraintes actives. Pour les plus gros problèmes, l'algorithme a du mal a trouver l'ensemble actif optimal : par exemple pour la matrice bande 1000 x 800 avec les contraintes [-1,1], il lui faut 210 itérations pour trouver 125 contraintes.

- Comme il a déjà été dit, on ne peut pas comparer les temps CPU pris par les deux méthodes sans certaines réserves. En effet, la méthode de Björck n'exploite pas le creux existant dans la plupart des matrices testées. De même, lorsque l'on teste les matrices pleines, le nouvel algorithme perd du temps avec un adressage par pointeurs. Ce qui donne une nette supériorité à la nouvelle méthode lorsque les matrices sont grandes et creuses, alors que la méthode de Björck va un peu plus vite dans le cas des matrices pleines.

- Lorsque l'on teste la matrice pleine 176 x 176 avec des bornes non significatives ($[-10^5, 10^5]$), on remarque que l'algorithme LSQR prend 300 itérations pour résoudre le problème. Pourtant, LSQR est un algorithme de gradients conjugués et devrait, théoriquement, prendre n itérations. On peut faire la même remarque pour la matrice 80 x 80, qui demande 143 itérations à LSQR pour résoudre le problème sans contrainte.
  On a calculé une estimation du conditionnement des différentes matrices pleines, par la méthode expliquée à la page 11. Voici les conditionnements obtenus :
  Matrice 100 x 50 : $8.29\ 10^1$
  Matrice 1000 x 5 : 5
  Matrice 80 x 80 : $4.39\ 10^3$
  Matrice 176 x 176 : $5.32\ 10^3$
  Rappelons que cette estimation est une borne inférieure du conditionnement.
  Nous ne savons pas expliquer le nombre, a priori important, d'itérations. Il est peut-être dû à un mauvais conditionnement du problème, bien que l'estimation donnée par LSQR ne le laisse pas penser. Ou bien le fait que l'algorithme LSQR donne, théoriquement, les mêmes itérés qu'un algorithme de gradients conjugués ne se vérifie-t-il pas dans la pratique ? Nous ne sommes pas parvenus à vérifier une de ces explications.

- Il est intéressant de regarder la proportion de temps CPU prise par l'algorithme LSQR par rapport au temps total de la nouvelle méthode. Lorsque, pour les grandes matrices, on place les bornes à [0,1], pratiquement tout le travail est effectué par les itérations majeures. Par exemple, pour la matrice bande 1000 x 800, LSQR prend 11 secondes sur les 7 minutes 17 secondes utilisées au total, ce qui fait environ 0.02 %, alors qu'il y a 432 variables actives

à la solution. Si on compare ces résultats avec ceux obtenus lorsque les bornes sont $[-10^5,0]$, on remarque que pour 378 variables actives à la solution LSQR prend 50 % du temps total. On peut expliquer ce phénomène par le fait que le domaine admissible est très étroit dans le premier cas, et que LSQR doit s'arrêter dès qu'il sort de ce domaine.

- La méthode de Björck est essentiellement basée sur la décomposition QR de la matrice A et sur la mise à jour de celle-ci. On remarque en fait que ce travail prend la majeure partie du temps d'éxécution. Et encore, on n'a pas tenu compte du temps pris pour la résolution du système triangulaire.

5.4 Comportement des algorithmes pour des problèmes mal conditionnés

Pour tester ce comportement, on va créer, à partir des matrices déjà construites, des matrices singulières et mal conditionnées.
Pour une matrice donnée, on prend sa première colonne à laquelle on ajoute une perturbation de la forme

$$\frac{\epsilon}{\|A_1\|}$$

où A_1 est la première colonne de la matrice et ϵ vaut successivement $0, 10^{-7}, 10^{-5}$ et 10^{-3}
Le vecteur ainsi obtenu va constituer la $(n+1)^{\text{ième}}$ colonne de la matrice.
De cette manière, si $\epsilon = 0$, on obtient donc une matrice où la première colonne est la même que la dernière, i.e. une matrice singulière. Par contre, si ϵ est petit mais non nul, on obtiendra une matrice mal conditionnée.

Pour ne pas multiplier les tests, nous nous sommes limités à une matrice aléatoire 1000 x 401, une matrice bande 1000 x 401 et une matrice pleine 80 x 81.
Les tableaux 13 à 24 donnent, pour chaque ϵ, les mêmes indications que dans les tableaux précedents.
Remarques :
Dans certains cas, les résultats de la nouvelle méthodes ne sont pas indiqués et ont été remplacés par des '?'. Il y a deux raisons à cela :

- soit l'algorithme a atteint 1000 itérations majeures (on l'a noté > 1000 dans la colonne It. Maj.). Cela est dû au fait que l'on n'atteint jamais, pour des raisons numériques, la précision désirée.

- soit le programme nécessite plus de trois heures de temps CPU, ce qui est la limite permise par la machine. Après trois heures, le programme s'arrête sans communiquer de résultats. Dans ce cas, nous avons remplacés ceux-ci par des '?'.

Bornes : $[-10^5, 10^5]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\epsilon = 0$	4	-	1	433.22	442.02
	$\epsilon = 10^{-7}$	2	-	1	434.21	433.58
	$\epsilon = 10^{-5}$	2	-	1	415.15	414.56
	$\epsilon = 10^{-3}$	2	-	1	461.49	460.84
Nouvelle méthode	$\epsilon = 0$	1	78	0	14.60	14.16
	$\epsilon = 10^{-7}$	> 1000	?	?	?	?
	$\epsilon = 10^{-5}$	> 1000	?	?	?	?
	$\epsilon = 10^{-3}$	> 1000	?	?	?	?

Table 13: Matrice aléatoire 1000 x 401

Bornes : $[-10^5, 0]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\epsilon = 0$	202	-	197	595.18	538.69
	$\epsilon = 10^{-7}$	200	-	197	578.98	522.90
	$\epsilon = 10^{-5}$	200	-	197	573.98	517.67
	$\epsilon = 10^{-3}$	200	-	197	614.41	556.65
Nouvelle méthode	$\epsilon = 0$	8	46	196	11.43	5.39
	$\epsilon = 10^{-7}$	10	86	197	17.43	10.47
	$\epsilon = 10^{-5}$	10	78	197	16.57	9.56
	$\epsilon = 10^{-3}$	10	69	197	15.99	8.90

Table 14: Matrice aléatoire 1000 x 401

Bornes : $[-1, 1]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\varepsilon = 0$	4	-	1	444.02	442.77
	$\varepsilon = 10^{-7}$	2	-	1	429.11	428.50
	$\varepsilon = 10^{-5}$	2	-	1	410.30	409.69
	$\varepsilon = 10^{-3}$	2	-	1	459.28	458.66
Nouvelle méthode	$\varepsilon = 0$	1	78	0	14.31	13.88
	$\varepsilon = 10^{-7}$	3	233	1	43.88	42.74
	$\varepsilon = 10^{-5}$	5	207	1	39.17	37.31
	$\varepsilon = 10^{-3}$	3	174	1	33.69	32.51

Table 15: Matrice aléatoire 1000 x 401

Bornes : $[0, 1]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\varepsilon = 0$	213	-	210	603.50	543.75
	$\varepsilon = 10^{-7}$	213	-	210	601.05	541.80
	$\varepsilon = 10^{-5}$	213	-	210	582.16	522.67
	$\varepsilon = 10^{-3}$	213	-	210	603.72	543.36
Nouvelle méthode	$\varepsilon = 0$	7	79	210	37.50	8.35
	$\varepsilon = 10^{-7}$	7	79	210	36.91	8.03
	$\varepsilon = 10^{-5}$	7	79	210	36.79	8.22
	$\varepsilon = 10^{-3}$	7	79	210	36.88	7.97

Table 16: Matrice aléatoire 1000 x 401

Bornes : $[-10^5, 10^5]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\varepsilon = 0$	2	-	1	177.58	177.06
	$\varepsilon = 10^{-7}$	2	-	1	179.35	178.81
	$\varepsilon = 10^{-5}$	2	-	1	183.16	182.63
	$\varepsilon = 10^{-3}$	2	-	1	185.31	184.77
Nouvelle méthode	$\varepsilon = 0$	1	748	0	117.17	116.76
	$\varepsilon = 10^{-7}$	?	?	?	?	?
	$\varepsilon = 10^{-5}$	?	?	?	?	?
	$\varepsilon = 10^{-3}$	?	?	?	?	?

Table 17: Matrice bande 1000 x 401

Bornes : $[-10^5, 0]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\varepsilon = 0$	242	-	209	343.43	279.25
	$\varepsilon = 10^{-7}$	242	-	209	343.06	279.26
	$\varepsilon = 10^{-5}$	243	-	209	339.92	276.83
	$\varepsilon = 10^{-3}$	243	-	209	349.55	285.04
Nouvelle méthode	$\varepsilon = 0$	15	156	209	25.80	14.83
	$\varepsilon = 10^{-7}$	15	156	209	26.49	14.83
	$\varepsilon = 10^{-5}$	15	156	209	26.98	15.00
	$\varepsilon = 10^{-3}$	15	156	209	27.07	15.11

Table 18: Matrice bande 1000 x 401

Bornes : $[-1, 1]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\epsilon = 0$	63	-	28	223.95	206.27
	$\epsilon = 10^{-7}$	61	-	28	221.00	204.16
	$\epsilon = 10^{-5}$	62	-	28	219.69	202.74
	$\epsilon = 10^{-3}$	62	-	28	219.71	202.14
Nouvelle méthode	$\epsilon = 0$	35	591	27	101.08	90.08
	$\epsilon = 10^{-7}$	38	1041	28	176.81	163.95
	$\epsilon = 10^{-5}$	38	981	28	166.60	153.86
	$\epsilon = 10^{-3}$	41	907	28	155.71	142.13

Table 19: Matrice bande 1000 x 401

Bornes : $[0, 1]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\epsilon = 0$	260	-	195	357.07	286.57
	$\epsilon = 10^{-7}$	260	-	195	353.97	285.36
	$\epsilon = 10^{-5}$	260	-	195	355.61	286.19
	$\epsilon = 10^{-3}$	260	-	195	359.91	290.79
Nouvelle méthode	$\epsilon = 0$	21	98	194	80.08	10.39
	$\epsilon = 10^{-7}$	22	1751	195	88.47	18.97
	$\epsilon = 10^{-5}$	22	160	195	88.86	17.40
	$\epsilon = 10^{-3}$	22	144	195	87.54	15.73

Table 20: Matrice bande 1000 x 401

Bornes : $[-10^5, 10^5]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\varepsilon = 0$	1	-	0	1.26	1.26
	$\varepsilon = 10^{-7}$	1	-	0	1.29	1.29
	$\varepsilon = 10^{-5}$	1	-	0	1.19	1.18
	$\varepsilon = 10^{-3}$	1	-	0	1.19	1.19
Nouvelle méthode	$\varepsilon = 0$	1	144	0	10.06	9.87
	$\varepsilon = 10^{-7}$	1	1471	0	10.48	10.29
	$\varepsilon = 10^{-5}$	1	145	0	10.74	10.51
	$\varepsilon = 10^{-3}$	1	143	0	10.02	9.84

Table 21: Matrice pleine 80 x 81

Bornes : $[-10^5, 0]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\varepsilon = 0$	52	-	41	3.09	2.29
	$\varepsilon = 10^{-7}$	52	-	41	3.31	2.43
	$\varepsilon = 10^{-5}$	52	-	41	3.02	2.23
	$\varepsilon = 10^{-3}$	52	-	41	3.07	2.24
Nouvelle méthode	$\varepsilon = 0$	6	50	40	3.27	2.05
	$\varepsilon = 10^{-7}$	11	1831	41	9.14	7.17
	$\varepsilon = 10^{-5}$	11	174	41	8.76	6.83
	$\varepsilon = 10^{-3}$	10	158	41	7.64	5.76

Table 22: Matrice pleine 80 x 81

Bornes : $[-1, 1]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\epsilon = 0$	21	-	4	1.91	1.52
	$\epsilon = 10^{-7}$	22	-	4	2.06	1.64
	$\epsilon = 10^{-5}$	21	-	4	1.89	1.52
	$\epsilon = 10^{-3}$	21	-	4	1.89	1.52
Nouvelle méthode	$\epsilon = 0$	14	425	3	31.02	29.01
	$\epsilon = 10^{-7}$	14	557	4	40.05	38.02
	$\epsilon = 10^{-5}$	13	522	4	37.99	36.09
	$\epsilon = 10^{-3}$	15	540	4	38.15	35.93

Table 23: Matrice pleine 80 x 81

Bornes : $[0, 1]$

		It. Maj.	It. Min.	Var. Act.	CPU	CPU (min)
Méthode de Björck	$\epsilon = 0$	68	-	39	3.66	2.56
	$\epsilon = 10^{-7}$	68	-	39	3.55	2.53
	$\epsilon = 10^{-5}$	68	-	39	3.59	2.54
	$\epsilon = 10^{-3}$	68	-	39	3.58	2.52
Nouvelle méthode	$\epsilon = 0$	9	59	39	4.06	2.46
	$\epsilon = 10^{-7}$	9	59	39	4.13	2.50
	$\epsilon = 10^{-5}$	9	59	39	4.25	2.60
	$\epsilon = 10^{-3}$	9	59	39	4.10	2.48

Table 24: Matrice pleine 80 x 81

Voici quelques commentaires sur ces résultats :

- La méthode de Björck semble plus sensible à des problèmes singuliers qu'à des problèmes mal conditionnés. En effet, le nombre d'itérations est souvent plus élevé lorsque le problème est singulier que lorsqu'il est mal conditionné. L'explication est simple : Björck suppose que la matrice définissant le problème est de rang plein lorsqu'il décrit son algorithme.

- On remarque également que lorsque l'on place les bornes à $[-10^5, 10^5]$, la méthode de Björck trouve une variable active à la solution. On peut supposer que si on n'avait pas mis de contraintes de bornes, la variable en question aurait continué à grandir en valeur absolue. Pour la matrice pleine (80 x 81) par contre il trouve une solution sans contraintes actives.

- En ce qui concerne la nouvelle méthode, en général elle prend moins d'itérations majeures lorsque le problème est singulier que lorsqu'il est mal conditionné.

- De même, si on ne regarde que l'algorithme LSQR (lorsque les bornes sont $[-10^5, 10^5]$), on remarque qu'il est très sensible à un mauvais conditionnement. Dans certains cas, il n'arrive pas à la précision désirée, ce qui provoque un bouclage de l'algorithme, qui s'arrête lorsqu'il atteint 1000 itérations ou lorsque le temps CPU utilisé atteint 3 heures. Mais même dans les cas où l'algorithme trouve une solution, on constate que pour le même nombre d'itérations majeures, plus le système est mal conditionné (i.e. plus ε est proche de 0), plus le nombre d'itérations mineures augmente. Par exemple, pour la matrice bande 1000 x 401 (bornes : [0,1]), pour 22 itérations majeures, on a respectivement 144 (pour $\varepsilon = 10^{-3}$),160 (pour $\varepsilon = 10^{-5}$) et 1751 (pour $\varepsilon = 10^{-7}$) itérations mineures.

5.5 Influence de la condition de complémentarité stricte

La théorie de convergence pour la nouvelle méthode (décrite dans [2]) nécessite la condition de complémentarité stricte (voir (48)). Nous allons tester le comportement de notre nouvel algorithme lorsque cette condition n'est pas vérifiée.
Pour ce faire, après avoir résolu un problème sans contrainte, on va placer les bornes juste à la solution obtenue. Pour voir ce qui se passe lorsque la condition est vérifiée, mais de peu (i.e. si $|(\nabla f)_i|$ est très petit mais non nul pour i dans l'ensemble actif) nous allons ensuite perturber un peu les bornes.
Quatre perturbations vont ainsi être faites :

$$\varepsilon = 10^{-5}$$
$$\varepsilon = 10^{-7}$$
$$\varepsilon = -10^{-7}$$
$$\varepsilon = -10^{-5}$$

La perturbation $\varepsilon = 0$ correspond au cas où la condition de complémentarité stricte (48) n'est pas vérifiée.
Lorsque la perturbation est négative, la solution sans contrainte est admissible, et lorsqu'elle est positive, on exclut cette solution sans contrainte du domaine admissible.
Il reste encore à déterminer le nombre de bornes que l'on va ainsi placer sur la solution sans contraintes.
Pour chaque exemple, nous avons choisi de prendre :

- d'abord une seule borne
- ensuite la moitié des bornes
- enfin toutes les bornes

Pour chaque test, nous donnons

- Le nombre d'itérations majeures (It. Maj.)
- Le nombre d'itérations mineures (It. Min.)
- Le nombre de variables actives à la solution (Var. Act.)

Remarque : nous donnons seulement les résultats des essais sur le nouvel algorithme. Les résultats obtenus avec la méthode de Björck n'étaient pas significatifs.

1 borne					
	Perturbation				
	$\varepsilon = -10^{-5}$	$\varepsilon = -10^{-7}$	$\varepsilon = 0$	$\varepsilon = 10^{-7}$	$\varepsilon = 10^{-5}$
It. Maj.	4	4	3	3	3
It. Min.	146	137	95	95	95
Var. Act.	0	0	1	1	1

200 bornes					
	Perturbation				
	$\varepsilon = -10^{-5}$	$\varepsilon = -10^{-7}$	$\varepsilon = 0$	$\varepsilon = 10^{-7}$	$\varepsilon = 10^{-5}$
It. Maj.	79	97	144	97	76
It. Min.	181	182	144	120	105
Var. Act.	0	0	86	196	196

400 bornes					
	Perturbation				
	$\varepsilon = -10^{-5}$	$\varepsilon = -10^{-7}$	$\varepsilon = 0$	$\varepsilon = 10^{-7}$	$\varepsilon = 10^{-5}$
It. Maj.	47	59	90	61	45
It. Min.	149	145	90	69	65
Var. Act.	0	0	174	376	376

Table 25: Matrice aléatoire 1000 x 400

1 borne					
	Perturbation				
	$\epsilon = -10^{-5}$	$\epsilon = -10^{-7}$	$\epsilon = 0$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-5}$
It. Maj.	2	2	4	2	2
It. Min.	1379	1379	1300	1390	1389
Var. Act.	0	0	1	1	1

200 bornes					
	Perturbation				
	$\epsilon = -10^{-5}$	$\epsilon = -10^{-7}$	$\epsilon = 0$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-5}$
It. Maj.	>1000	>1000	>1000	>1000	>1000
It. Min.	?	?	?	?	?
Var. Act.	?	?	?	?	?

400 bornes					
	Perturbation				
	$\epsilon = -10^{-5}$	$\epsilon = -10^{-7}$	$\epsilon = 0$	$\epsilon = 10^{-7}$	$\epsilon = 10^{-5}$
It. Maj.	113	163	208	121	110
It. Min.	1264	2239	210	157	131
Var. Act.	0	0	180	330	330

Table 26: Matrice bande 1000 x 400

1 borne					
	Perturbation				
	$\varepsilon = -10^{-5}$	$\varepsilon = -10^{-7}$	$\varepsilon = 0$	$\varepsilon = 10^{-7}$	$\varepsilon = 10^{-5}$
It. Maj.	2	2	3	7	8
It. Min.	251	251	370	519	510
Var. Act.	0	0	0	1	1

40 bornes					
	Perturbation				
	$\varepsilon = -10^{-5}$	$\varepsilon = -10^{-7}$	$\varepsilon = 0$	$\varepsilon = 10^{-7}$	$\varepsilon = 10^{-5}$
It. Maj.	123	118	168	128	98
It. Min.	901	493	299	282	236
Var. Act.	0	0	18	27	27

80 bornes					
	Perturbation				
	$\varepsilon = -10^{-5}$	$\varepsilon = -10^{-7}$	$\varepsilon = 0$	$\varepsilon = 10^{-7}$	$\varepsilon = 10^{-5}$
It. Maj.	96	110	150	39	36
It. Min.	674	783	195	49	74
Var. Act.	0	0	31	80	62

Table 27: Matrice pleine 80 x 80

La théorie développée par Conn, Gould et Toint dans [2] affirme que lorsque la condition de complémentarité stricte est vérifiée, l'algorithme trouvera le bon ensemble actif en un nombre fini d'itérations. Mais parviendra-t-il à le trouver lorsque cette condition n'est pas vérifiée ?
On voit nettement que dans tous les cas sauf un, l'absence de la condition de complémentarité stricte n'empêche pas l'algorithme de converger, même s'il lui faut plus d'itérations pour cela.
Pour les trois matrices testées, on constate que l'algorithme éprouve plus de difficultés à trouver la solution lorsque l'on ne place que la moitié des bornes sur la solution sans contrainte, que lorsqu'on les place toutes. Dans le cas des matrices bandes, l'algorithme ne converge toujours pas après 1000 itérations.
Il est à remarquer également que lorsque les bornes se trouvent très proches de la solution sans contrainte, mais sans l'exclure ($\varepsilon = -10^{-5}$ et $\varepsilon = -10^{-7}$), l'algorithme doit tout de même prendre plusieurs itérations majeures. En effet, on limite le rayon d'action du LSQR, en ce sens qu'il ne peut pas trouver la solution sans contrainte à la première itération majeure, car on l'arrête dès qu'un itéré viole une borne. Si on décidait de laisser aller LSQR jusqu'au bout, il trouverait tout de suite la solution sans contrainte, sans se soucier des bornes. Malheureusement, cette méthode n'est pas rentable dans les cas généraux : il est tout à fait inutile de résoudre les sous-problèmes complètement lorsqu'on n'a pas le bon ensemble actif.

6 Perspectives et conclusion

- La méthode telle qu'elle est présentée ici est une première proposition, en ce sens qu'elle peut certainement être améliorée. Voici quelques exemples d'améliorations possibles :

  1. Pour augmenter la rapidité de convergence de l'algorithme LSQR, on pourrait étudier la possibilité d'introduire une matrice de préconditionnement. Quelle matrice choisir ? Comment l'introduire dans l'algorithme de Paige et Saunders ? Ce sont des questions ouvertes. Une possibilité assez simple pour le préconditionneur serait de choisir une matrice diagonale dont les éléments seraient la norme des différentes colonnes de la matrice A

     $$P = \text{diag}(\|a_1\|, \|a_2\|, \ldots, \|a_n\|)$$

     où P est la matrice de préconditionnement et a_i la $i^{\text{ième}}$ colonne de A
  2. Le calcul du point de Cauchy généralisé est en fait une recherche linéaire (exacte) le long d'une ligne brisée. On pourrait imaginer de remplacer cette recherche linéaire exacte par une recherche linéaire inexacte (voir [13], [15] et [16]).

- Pour pouvoir réellement tester la qualité de la nouvelle stratégie de contraintes actives, il faudrait :

  - ou bien utiliser la factorisation QR pour résoudre les problèmes sans contrainte sur les variables libres dans la nouvelle méthode , et faire la comparaison avec la méthode de Björck
  - ou bien transformer la méthode de Björck pour qu'elle résolve les sous-problèmes avec l'algorithme LSQR, comme l'a suggéré Lötstedt, et la comparer avec notre algorithme.

Ainsi, on aurait pour chaque méthode de contraintes actives un même algorithme résolvant les sous-problèmes sans contrainte.

- On pourrait généraliser cette méthode de contraintes actives au cas de contraintes linéaires plus générales. Pour cela, il faudrait trouver un bon opérateur de projection sur le domaine admissible. Cet opérateur servirait à déterminer les différentes directions de recherche pour le calcul du point de Cauchy généralisé. Dans le cas de contraintes de bornes que nous avons traité, il suffisait d'annuler certaines composantes de la direction de recherche initiale pour obtenir les différents arcs (voir section 3.4). Cela devient moins simple lorsqu'il s'agit de contraintes linéaires quelconques.
  Cela vaudrait tout de même la peine de s'y intéresser, vu la rapidité de cette stratégie à détecter l'ensemble actif à la solution.

En conclusion, nous estimons que cette méthode est tout à fait digne d'intérêt par sa rapidité à détecter l'ensemble actif optimal, et mérite d'être exploitée.

7 Remerciements

Je tiens à remercier tout spécialement M. Ph. Toint et M. D. Tuyttens qui m'ont permis, par l'intérêt qu'ils portaient à ce travail, de le mener à bien.

References

[1] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 1983

[2] A.R. Conn, N.I.M. Gould and Ph. L. Toint, *Global convergence of a class of trust region algorithms for optimization with simple bounds*, Technical Report 86/1, Dept. of Maths, FUN, Namur, 1986, to appear, SIAM Journal on Numerical Analysis, 1987.

[3] A.R Conn, N.I.M. Gould and Ph. L. Toint, *Testing a class of methods for solving minimization problems with simple bounds on the variables*, 1987

[4] C. C. Paige and M. A. Saunders, *LSQR : An Algorithm for Sparse Linear Equations and Sparse Least Squares*,ACM Transactions on Mathematical Software, Vol. 8 No 1, March 1982, pages 43-71.

[5] M. R. Hestenes and E. Stiefel *Methods of Conjugate Gradients for Solving Linear Systems*, J. Res. Nat. Bur. Stand. 49, 409-36, 1952

[6] Åke Björck, *Least Squares Methods*,Department of Mathematics, Linköping University, S-581 83 Linköping, SWEDEN. To appear in : *Handbook of Numerical Analysis* Vol. 1 : Solution of Equations in $\mathbf{R}^n$, Ciarlet and Lions, Elsevier, North Holland, 1987.

[7] Per Lötstedt, *Solving the minimal least squares problem subject to bounds on the variables*, BIT 24, pp. 206-224

[8] J.J. Dongarra, C.B. Moler, J.R. Bunch, G.W. Stewart, *LINPACK Users' Guide*, SIAM, Philadelphia 1979

[9] M.Z. Nashed, *Generalized Inverses and Applications*, Academic Press, New York, 1976

[10] M. Gentleman, *Least Squares Computations by Givens Transformations without Square Roots*, J. Inst. Math. Appl. 12, p. 329-336, 1973

[11] S. Hammarling, *A Note on Modifications to the Givens Plane Rotation*, J. Inst. Math. Appl. 13, p. 215-218, 1974

[12] J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965

[13] J. E. Dennis and R.B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, New Jersey, 1983

[14] L. Ulrix, *Méthodes de moindres carrés linéaires avec contraintes linéaires. Application aux bilans cohérents*. Mémoire de licence en mathématiques, FNDP Namur, 1987

[15] Ph. L. Toint, *Global convergence of a class of trust region methods for nonconvex minimization in Hilbert space*,University of Namur, Department of Mathematics, Namur, Belgium, 1987. To appear : IMA, Journal of Numerical Analysis, 1988

[16] J. J. Moré, *Trust regions and projected gradients*, Mathematics and Computer Science Division, Argonne National Laboratory, University of Chicago, 1988.

Table des matières

Annexe

Voici en annexe le code des différents programmes utilisés pour les tests numériques.

- D'abord les routines de résolution de problèmes des moindres carrés avec contraintes de bornes
- Ensuite les programmes appelant ces routines
- Enfin les autres routines appelées par ces programmes

```
      SUBROUTINE MCARRE(A,INDLIGNE,DEBCOL,
     1                  B,
     1                  M,N,
     1                  L,U,
     1                  XK,
     1                  ATOL,BTOL,CONLIM,XNLIM,
     1                  US,IMPR,
     1                  EPSILON,
     1                  MAXITE,
     1                  RESLSQR)
```

Cette routine permet de resoudre le probleme aux moindres carres avec contraintes de bornes.

On veut resoudre :

$$\begin{cases} \min \|Ax-b\| \\ \text{s.c. } L(i) \leq X(i) \leq U(i) \end{cases}$$

ou ||.|| est la norme 2 et A est une matrice creuse

Michel BIERLAIRE

2de licence mathematique

Routines utilisees

Dans la librairie ANLIB :

DCMACH,DSETVL

Dans la librairie UTIL :

ISECOND

Dans la librairie BLAS :

DCOPY,DAXPY

Dans le fichier ROUTINES :

ADM,RSCSP,GRAD,RECHERCHE,CASSURE,COLONNE

```
C       Dans le fichier MICLSQR :
C       -------------------------
C
C
C       CLSQR2
C
C       ==============
C       | Parametres |
C       ==============
C
        INTEGER NMAX
        PARAMETER (NMAX=1000)
C
C               dimension maximale de X
C
        INTEGER MMAX
        PARAMETER (MMAX=1000)
C
C               dimension maximale de B
C
        INTEGER NZEROMAX
        PARAMETER (NZEROMAX=31000)
C
C               nombre maximal d'elements non nuls dans la matrice creuse A
C
C       =============
C       | Arguments |
C       =============
C
        DOUBLE PRECISION A(NZEROMAX)
C
C           INPUT :
C                vecteur contenant les elements non nuls de la matrice A
C                stockes colonne par colonne
C           OUTPUT :
C                inchange
C
        INTEGER INDLIGNE(NZEROMAX)
C
C           INPUT :
C                vecteur contenant les indices lignes des elements non nuls
C                de A colonne par colonne
C           OUTPUT :
C                inchange
```

```
      INTEGER DEBCOL(NMAX+1)
C
C        INPUT :
C             vecteur contenant les pointeurs vers les premiers elements
C             non nuls des colonnes successives dans INDLIGNE et A.
C             DEBCOL(i) contient la position dans A et INDLIGNE du premier
C             element de la ligne no i. DEBCOL(N+1) contient la longueur
C             effective des vecteurs A et DEBLIGNE
C        OUTPUT :
C             inchange
C
      DOUBLE PRECISION B(MMAX)
C
C        INPUT :
C             vecteur contenant le membre de droite du probleme
C        OUTPUT :
C             inchange
C
      INTEGER M
C
C        INPUT :
C             dimension effective de B et nombre de lignes de A
C        OUTPUT :
C             inchange
C
      INTEGER N
C
C        INPUT :
C             dimension effective de X et nombre de colonnes de A
C        OUTPUT :
C             inchange
C
      DOUBLE PRECISION L(NMAX),U(NMAX)
C
C        INPUT :
C             vecteurs contenant les bornes imposees au variables
C             L contient les bornes inferieures
C             U contient les bornes superieures
C        OUTPUT :
C             inchange
C
      DOUBLE PRECISION XK(NMAX)
C
C        INPUT :
C             sans importance
C        OUTPUT :
C             solution du probleme (du moins on l'espere !)
C
      DOUBLE PRECISION ATOL
C
C        INPUT :
C             estimation de l'erreur relative sur les donnees definissant
C             la matrice A
C        OUTPUT :
C             inchange
C
```

```
C
      DOUBLE PRECISION BTOL
C
C        INPUT :
C             estimation de l'erreur relative sur les donnees definissant
C             le vecteur B
C        OUTPUT :
C             inchange
C
      DOUBLE PRECISION CONLIM
C
C        INPUT :
C             borne superieure sur le conditionnement de A
C        OUTPUT :
C             inchange
C
      DOUBLE PRECISION XNLIM
C
C        INPUT :
C             borne superieure sur la norme 2 de la solution X
C        OUTPUT :
C             inchange
C
      INTEGER US
C
C        INPUT :
C             numero de l'unite de sortie des impressions
C        OUTPUT :
C             inchange
C
      INTEGER IMPR
C
C        INPUT :
C             < 0 si on ne desire aucune information
C             = 0 si on desire uniquement les informations finales
C             > 0 si ondesire des informations toutes les IMPR iterations
C        OUTPUT :
C             inchange
C
      DOUBLE PRECISION EPSILON
C
C        INPUT :
C             precision desiree
C        OUTPUT :
C             inchange
C
      INTEGER MAXITE
C
C        INPUT :
C             nombre maximum d'iterations
C        OUTPUT :
C             inchange
```

```
      INTEGER RESLSQR

C
C        INPUT :
C             indique ce qu'on doit sortir comme resultat pour LSQR
C             Si < 0  pas de resultat
C             Si = 0  resultat a la premiere et la derniere iteration
C             Si > 0  resultat toutes les RESLSQR iterations
C        OUTPUT :
C             inchange
C
C     ==============
C     ! Fonctions !
C     ==============
C
      DOUBLE PRECISION DDOT,CDDOT,DNRM2,MINVEC
      LOGICAL CAL,OPTIMAL
      INTEGER ISECONDIFF
      EXTERNAL DDOT,CDDOT,DNRM2,MINVEC,CAL,ISECONDIFF,OPTIMAL
      INTRINSIC ABS
C
C     ============================
C     ! Variables de la routine !
C     ============================
C
      DOUBLE PRECISION AI(MMAX)
C
C             contient la I ieme colonne de A
C
      DOUBLE PRECISION MCEPS
C
C             precision de la machine
C
      DOUBLE PRECISION GRAND,PETIT
C
C             les limites d'overflow et d'underflow de la machine
C
      DOUBLE PRECISION ZERO,UN
C
C             contiennent respectivement les valeurs 0.0d0 et 1.0d0
C
      INTEGER COMPTG
C
C             nombre d'evaluations du gradient
C
      INTEGER ITEMAJ
C
C             nombre d'iterations majeures
C
      INTEGER ITEMIN
C
C             nombre total d'iterations mineures
C
      INTEGER NOMBACT
C
C             nombre de variables actives a la solution
```

```
      INTEGER CPU
C
C           temps CPU utilise par la routine
C
      INTEGER TOTLSQR,CPULSQR
C
C           temps CPU utilise par LSQR
C
      INTEGER ITERCAUCHY
C
C           nombre d'iterations pour trouver le PCG
C
      LOGICAL OK
C
C           .TRUE. si on a un point admissible
C
      LOGICAL ECRIRE
C
C           .TRUE. si on veut des impressions lors de l'iteration en cours
C
      DOUBLE PRECISION GK(NMAX)
C
C           gradient de la fonction en XK
C
      DOUBLE PRECISION ATB(NMAX)
C                           t
C           valeur de (- A B) calculee une fois pour toutes
C
      DOUBLE PRECISION D(NMAX)
C
C           utilise dans la recherche du point de Cauchy generalise
C
      DOUBLE PRECISION BB(NMAX)
C
C           utilise dans la recherche du point de Cauchy generalise (PCG)
C
      DOUBLE PRECISION F1,F2
C
C           valeur des derivees dans la recherche du PCG.
C
      DOUBLE PRECISION Q
C
C           - quotient de F1 et F2
C
      DOUBLE PRECISION XKC(NMAX)
C
C           point de Cauchy generalise
C
      DOUBLE PRECISION YM(MMAX)
C
C           vecteur de travail
C
      DOUBLE PRECISION T11,T12,T21,T22
C
C           variables intermediaires
```

```
      INTEGER CALEE(NMAX)

C
C             CALEE(i) = 0 si XKC(i) est une variable libre
C             CALEE(i) = 1 si XKC(i) atteint sa borne inferieure
C             CALEE(i) = 3 si XKC(i) atteint sa borne superieure
C
      INTEGER ITLSQR

C
C             nombre d'iterations effectuees par LSQR
C
      INTEGER MAXLSQR

C
C             nombre d'iterations maximum pour LSQR
C
      DOUBLE PRECISION RNORM

C
C             evaluation de la norme du residu de LSQR
C
      DOUBLE PRECISION APCOND

C
C             estimation du nombre de conditionement de la matrice reduite
C
      DOUBLE PRECISION COND

C
C             conditionnement le plus grand parmi celui de toutes les sous-
C             matrices
C
      DOUBLE PRECISION XPNORM

C
C             estimation de la norme de la solution de LSQR
C
      INTEGER STOPLSQR

C
C             raison de l'arret de LSQR :
C             1. On a trouve la solution exacte
C             2. On a depasse le nombre maximum d'iterations
C             3. On a trouve une solution des moindres carres
C             4. L'estimation du conditionnement a depasse la borne permise
C             5. La norme de la solution a depasse la borne permise
C             6. On est sorti du domaine admissible
C
      DOUBLE PRECISION WRK,WORK(2*NMAX)

C             vecteurs de travail de LSQR
C
      DOUBLE PRECISION APNORM

C             estimation de la norme de Frobenius de AP
C
      DOUBLE PRECISION ARNORM

C             estimation de la norme du gradient de la norme du residu final
```

```
      DOUBLE PRECISION BTRANSLATE(MMAX)
C
C           contient le vecteur B translate (pour demarrer LSQR avec
C           0 admissible)
C
      DOUBLE PRECISION UTRANSLATE(NMAX)
C
C           contient les bornes superieures du probleme translate
C
      DOUBLE PRECISION LTRANSLATE(NMAX)
C
C           contient les bornes inferieures du probleme translate
C
      INTEGER I,J
C
C           indices de boucles
C
      DOUBLE PRECISION DELT
C
C           pas le plus grand que l'on peut faire le long d'une direction
C           tout en restant admissible.
C
      LOGICAL SUP
C
C           .TRUE. si on se trouve a une borne superieure
C
      LOGICAL INF
C
C           .TRUE. si on se trouve a une borne inferieure
C
C
C     ==============
C     ! Programme !
C     ==============
C
C     Initialisations :
C     ================
C
C     Calcul des caracteristiques de la machine
C     -----------------------------------------
C
      CALL DCMACH(MCEPS,GRAND,PETIT,ZERO,UN)
C
C     Compteur de gradient
C     --------------------
C
      COMPTG=0
C
C     Compteur d'iterations majeures
C     ------------------------------
C
      ITEMAJ=0
```

```
C
C       Compteur d'iterations mineures
C       ------------------------------
C
C
        ITEMIN=0
C
C       Initialisation pour le calcul du temps CPU
C       ------------------------------------------
C
C
        CALL ISECOND(CPU)
C
C       Pour LSQR
C       ---------
C
C
        MAXLSQR=5000
C
C       Affichage
C       ---------
C
C
        IF (IMPR.GE.0) THEN
           WRITE(US,1000)
 1000      FORMAT(/,/,' Resultats de l''algorithme MCARRE pour la '
     1          ,'resolution de problemes des moindres carres avec'
     2          ,' contraintes de bornes.',/,116('='),/)
           WRITE(US,1110)M,N
 1110      FORMAT(' Matrice A (',I4,'x',I4,')')
        ENDIF
C
C       Choix de XO
C       -----------
C
C
        CALL ADM(XK,N,L,U,OK,ZERO)
C                    t
C       Calcul de A B
C       -------------
C
C
C       ATB=0
C
        CALL DSETVL (N,ATB,1,ZERO)
C                    t
C       ATB=ATB - A B
C
        CALL RSCSP(M,N,A,B,ATB,.TRUE.,.FALSE.,INDLIGNE,DEBCOL)
C
C       Calcul du gradient en XO
C       ------------------------
C
C
        CALL DCOPY(N,ATB,1,GK,1)
        IF (.NOT.OK) THEN
           CALL GRAD(M,N,XK,A,GK,INDLIGNE,DEBCOL)
           COMPTG=COMPTG+1
        ENDIF
```

```
C        Debut des iterations
C        ====================
C
     1   ITEMAJ=ITEMAJ+1
         IF (ITEMAJ.GT.MAXITE) THEN
                 IF (IMPR.GE.0) WRITE(US,1005)MAXITE
 1005            FORMAT(/,' L''algorithme a atteint les ',I3,
         1               ' iterations permises.')
                 RETURN
         ENDIF
         IF (IMPR.EQ.0) THEN
            ECRIRE=.FALSE.
         ELSE
            ECRIRE=(MOD(ITEMAJ,IMPR).EQ.0)
         ENDIF
         IF (ECRIRE) WRITE(US,1010)ITEMAJ
 1010    FORMAT(/,' Iteration ',I3,/,14('-'))
C
C        Test de convergence
C        ===================
C
         IF (OPTIMAL(XK,GK,N,EPSILON,L,U,ECRIRE,US)) THEN
                 ITEMAJ=ITEMAJ-1
                 IF (IMPR.GE.0) THEN
                    CPU=ISECONDIFF(CPU)
                    IF (N.LE.20) THEN
                          WRITE(US,10)(I,XK(I),I=1,N)
    10                    FORMAT(/,4(1X,'XK(',I3,')=',FD14.7,5X))
                          WRITE(US,*)('est accepte comme solution ')
                    ENDIF
                    WRITE(US,11)ITEMAJ,ITEMIN,CPU,COMPTG,NOMBACT,TOTLSQR
         1                              ,COND
    11                   FORMAT(' Iterations majeures :              ',I3,/,
         1                 ' Iterations mineures :          ',I7,/,
         2                 ' Temps CPU :                    ',I7,/,
         3                 ' Evaluations du gradient :        ',I3,/,
         4                 ' Nombre de variables actives :    ',I3,/,
         5                 ' CPU pour LSQR :                ',I7,/,
         6                 ' Conditionnement :              ',1PD15.7)
                 ENDIF
                 RETURN
         ENDIF
C
C        Recherche du point de Cauchy generalise
C        =======================================
C
C        Initialisations
C        ---------------
C
         ITERCAUCHY=0
         CALL DCOPY(N,XK,1,XKC,1)
```

```
C
C       direction de recherche
C
        CALL RECHERCHE(XKC,N,CALEE,GK,D,L,U,MCEPS,NOMBACT)
C
        F1=DDOT(N,GK,1,D,1)
        CALL DSETVL(M,YM,1,ZERO)
C
C       Y=Ad
C
        CALL RSCSP(M,N,A,D,YM,.FALSE.,.FALSE.,INDLIGNE,DEBCOL)
C
        F2=DDOT(M,YM,1,YM,1)
        IF (F1.GE.0) THEN
                GOTO 100
        ENDIF
C
C       Trouver le point de cassure suivant
C       -----------------------------------
C
    5   CONTINUE
        ITERCAUCHY=ITERCAUCHY+1
        CALL CASSURE(D,XKC,L,U,MCEPS,GRAND,DELT,N,CALEE,.TRUE.)
C
C       Test verifiant si on a trouve le PCG
C       ------------------------------------
C
        Q=-F1/F2
        IF ((F2.GT.0).AND.(Q.GT.0).AND.(Q.LT.DELT)) THEN
C
C               On a trouve le PCG
C
C               XKC=XKC+Q*D
C
                CALL DAXPY(N,Q,D,1,XKC,1)
                DO 1232 I=1,N
                   SUP=(DABS(XKC(I)-U(I)).LT.MCEPS)
                   INF=(DABS(XKC(I)-L(I)).LT.MCEPS)
                   IF (CALEE(I).EQ.0) THEN
                         IF (INF) THEN
                            XKC(I)=L(I)
                            CALEE(I)=1
                            NOMBACT=NOMBACT+1
                         ELSE IF (SUP) THEN
                            XKC(I)=U(I)
                            CALEE(I)=3
                            NOMBACT=NOMBACT+1
                         ENDIF
                   ELSE
                         IF (.NOT.INF .AND. .NOT. SUP) THEN
                            CALEE(I)=0
                            NOMBACT=NOMBACT-1
                         ENDIF
                   ENDIF
 1232           CONTINUE
                GOTO 100
        ENDIF
```

```
C
C       Mises a jour
C       ------------
C
C
C       XKC=XKC+DELT*D
C
        CALL DAXPY(N,DELT,D,1,XKC,1)
C                          t              i
C       Calcul de BB = A A (sum (d .e )  avec i indices des variables actives
C                            i     i
C
        CALL DSETVL(M,YM,1,ZERO)
        DO 40 I=1,N
           IF ((DABS(XKC(I)-L(I)).LT.MCEPS).OR.(DABS(XKC(I)-U(I))
     1                                .LT.MCEPS).AND. CALEE(I).EQ.0) THEN
              CALL COLONNE(A,INDLIGNE,DEBCOL,AI,M,I)
              CALL DAXPY(M,D(I),AI,1,YM,1)
           ENDIF
   40   CONTINUE
        CALL DSETVL(N,BB,1,ZERO)
        CALL RSCSP(M,N,A,YM,BB,.TRUE.,.TRUE.,INDLIGNE,DEBCOL)
C
        T11=ZERO
        T12=ZERO
        T21=ZERO
        T22=ZERO
        DO 45 I=1,N
           T21=T21+BB(I)*XKC(I)
           T22=T22+BB(I)*D(I)
           IF (CALEE(I).EQ.0) THEN
                 IF (DABS(XKC(I)-L(I)).LT.MCEPS) THEN
                    CALEE(I)=1
                    XKC(I)=L(I)
                    NOMBACT=NOMBACT+1
                    T11=T11+ATB(I)*D(I)
                    T12=T12+D(I)*BB(I)
                    D(I)=ZERO
                 ELSE IF (DABS(XKC(I)-U(I)).LT.MCEPS) THEN
                    CALEE(I)=3
                    XKC(I)=U(I)
                    NOMBACT=NOMBACT+1
                    T11=T11+ATB(I)*D(I)
                    T12=T12+D(I)*BB(I)
                    D(I)=ZERO
                 ENDIF
           ELSE
                 IF (XKC(I).NE.U(I) .AND. XKC(I).NE.L(I)) THEN
                    CALEE(I)=0
                    NOMBACT=NOMBACT-1
                 ENDIF
           ENDIF
   45   CONTINUE
        F1=F1+DELT*F2-T11-T21
        F2=F2+T12-2*T22
        IF (F1.LT.ZERO) GOTO 5
```

```
C
C        On a trouve le point de Cauchy generalise
C        -----------------------------------------
C
  100    IF (ECRIRE) WRITE(US,1030) ITERCAUCHY
 1030    FORMAT(/,' On a trouve le point de Cauchy en ',I3,' iterations ')
         IF ((N.LE.20).AND.(ECRIRE)) THEN
            WRITE(US,1040) (I,XKC(I),I=1,N)
 1040       FORMAT(/,' Le point de Cauchy generalise : ',/,(1X,'XC(',
     1         I3,') = ',PD15.7))
         ENDIF
         IF (ECRIRE) WRITE(US,1062)NOMBACT
 1062    FORMAT(' Nombre de variables actives : ',I3)
C
C        Cas ou toutes les variables sont actives
C        ----------------------------------------
C
         IF (NOMBACT.EQ.N) THEN
C
C                XK=XKC
C
                 CALL DCOPY (N,XKC,1,XK,1)
C
C                On affiche le nouvel itere
C
                 GOTO 1076
         ENDIF
C
C        Trouver le nouvel itere
C        =======================
C
C        A ce moment CALEE contient l'etat de l'activite des composantes du PCG
C
         On applique l'algorithme LSQR dans le sous espace
         -------------------------------------------------
C
C        Btranslate = B - A.XKC
C
         CALL DCOPY(M,B,1,BTRANSLATE,1)
         CALL RSCSP(M,N,A,XKC,BTRANSLATE,.FALSE.,.FALSE.,INDLIGNE,DEBCOL)
C
C        Utranslate = U - XKC
C
         CALL DCOPY(N,U,1,UTRANSLATE,1)
         CALL DAXPY(N,-UN,XKC,1,UTRANSLATE,1)
C
C        Ltranslate = L - XKC
C
         CALL DCOPY(N,L,1,LTRANSLATE,1)
         CALL DAXPY(N,-UN,XKC,1,LTRANSLATE,1)
         CPULSQR=ISECONDIFF(CPU)
         CALL CLSQR2(M,N,A,XK,BTRANSLATE,INDLIGNE,DEBCOL,
         1         ITLSQR,MAXLSQR,RNORM,ATOL,BTOL,APCOND,CONLIM,XPNORM,
         1         XNLIM,STOPLSQR,WORK,RESLSQR,US,APNORM,ARNORM,
         1         LTRANSLATE,UTRANSLATE,CALEE,WRK)
```

```
      COND=MAX(APCOND,COND)
      TOTLSQR=TOTLSQR+ISECONDIFF(CPU)-CPULSQR
      ITEMIN=ITEMIN+ITLSQR
      IF (ECRIRE) WRITE(US,1100)STOPLSQR
 1100    FORMAT(' Raison de l''arret du LSQR : ',I1)
      CALL DAXPY(N,UN,XKC,1,XK,1)
      IF ((STOPLSQR.EQ.4).OR.(STOPLSQR.EQ.5)) THEN
         IF (IMPR.GE.0) WRITE(US,80)STOPLSQR
   80          FORMAT(' L''algorithme LSQR a du etre interrompu :',I1)
         RETURN
      ENDIF
C
C     Calcul du nouvel itere
C     ----------------------
C
      IF (STOPLSQR.EQ.6) THEN
         DO 83 I=1,N
              D(I)=XK(I)-XKC(I)
   83    CONTINUE
         CALL CASSURE(D,XKC,L,U,MCEPS,GRAND,DELT,N,CALEE,.FALSE.)
      ENDIF
      DO 90 I=1,N
         IF (CALEE(I).NE.0) THEN
              XK(I)=XKC(I)
         ELSE IF (STOPLSQR.EQ.6) THEN
              XK(I)=XKC(I)+DELT*D(I)
         ENDIF
   90 CONTINUE
C
C     Calcul du gradient
C     ------------------
C
      CALL DCOPY(N,ATB,1,GK,1)
      CALL GRAD(M,N,XK,A,GK,INDLIGNE,DEBCOL)
      COMPTG=COMPTG+1
C
C     Affichage
C     ---------
C
 1076 IF ((N.LE.20).AND.(ECRIRE)) THEN
         WRITE(US,1080) (I,XK(I),I=1,N)
 1080    FORMAT(/,' Le nouvel itere : ',/,(1X,'X(',I3,') = ',PD15.7))
         WRITE(US,1200) (I,GK(I),I=1,N)
 1200    FORMAT(/,' Le gradient : ',/,(1X,'Grad(',I3,') = ',PD15.7))
      ENDIF
      GOTO 1
      END
```

```
      SUBROUTINE BJORCK(A,M,N,X,B,L,U,MAXITE,US)
C     ~~~~~~~~~~~~~~~~~~
C
C     Cette routine va resoudre le probleme des moindres carres suivant :
C
C
C                min   ||Ax-b||
C                 x             2
C
C                s.c    l <= x <= u
C
C     en utilisant l'algorithme decrit par Bjorck dans
C
C     'Least Squares Methods', to appear in HANDBOOK OF NUMERICAL ANALYSIS,
C     vol 1: solution of equations in Rn , Ciarlet and Lions, Elsevier,
C     North Holland,1987.
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire ,1988
C
C     ======================
C     | Routines utilisees |
C     ======================
C
C     Dans la librairie ANLIB :
C     -------------------------
C
C     DCMACH
C
C     Dans la librairie UTIL :
C     ------------------------
C
C     ISECOND
C
C     Dans la librairie LINPACK :
C     ---------------------------
C
C     DQRDC,DQRSL,DTRSL,DCHEX
C
C     Dans la librairie BLAS :
C     ------------------------
C
C     DCOPY,DAXPY
C
C     Dans le fichier ROUTINES :
C     --------------------------
C
C     PERM
```

```
C       ===============
C       | Parametres |
C       ===============
C
        INTEGER MMAX,NMAX
        PARAMETER (NMAX=1000,MMAX=1000)
C
C       ==============
C       | Arguments |
C       ==============
C
        DOUBLE PRECISION A(MMAX,NMAX)
C
C           INPUT :
C                matrice definissant le probleme
C           OUTPUT :
C                resultat de la factorisation QR et des differentes mises a jour
C
        INTEGER M,N
C
C           INPUT :
C                dimensions effectives de la matrice A
C           OUTPUT :
C                inchange
C
        DOUBLE PRECISION X(NMAX)
C
C           INPUT :
C                sans importance
C           OUTPUT :
C                solution du probleme si l'algorithme est arrive a son terme
C
        DOUBLE PRECISION B(MMAX)
C
C           INPUT :
C                membre de droite du probleme
C           OUTPUT :
C                inchange
C
        DOUBLE PRECISION L(NMAX),U(NMAX)
C
C           INPUT :
C                vecteurs contenant les bornes du probleme
C           OUTPUT :
C                inchange
C
        INTEGER MAXITE
C
C           INPUT :
C                nombre maximum d'iterations
C           OUTPUT :
C                inchange
```

```
      INTEGER US

C
C        INPUT :
C             numero de l'unite logique de sortie
C        OUTPUT :
C             inchange
C
C     ============================
C     ! Variables de la routine !
C     ============================
C
      INTEGER ACT(NMAX)

C             ACT(i) = 0  si x(i) est une variable libre
C                    = 1  si x(i) est fixee (borne inferieure)
C                    = -1 si x(i) est fixee (borne superieure)

      INTEGER FF,BB

C             le nombre de variables libres et de variables fixees (resp.)

      INTEGER I,J,K

C             variables de boucle

      DOUBLE PRECISION QRAUX(NMAX)

C             utilise par LINPACK pour les renseignements sur QR

      INTEGER JPVT(NMAX)

C             utilise par LINPACK pour les pivotages

      DOUBLE PRECISION QTB(MMAX)
                                      t
C             contient le produit Q b, ou Q est le resultat de la
C             factorisation QR de A.

      DOUBLE PRECISION WORK(NMAX)

C             vecteur de travail pour LINPACK

      DOUBLE PRECISION DUMMY0(MMAX)
      DOUBLE PRECISION DUMMY1(MMAX)
      DOUBLE PRECISION DUMMY2(NMAX)
      DOUBLE PRECISION DUMMY3(MMAX)
      DOUBLE PRECISION DUMMY4(MMAX)

C             arguments pour les routines de LINPACK, inutiles pour nous

      INTEGER INFO

C             information fournie par LINPACK
```

```
      INTEGER P(NMAX)
C
C           vecteur de permutations
C
      DOUBLE PRECISION XB(NMAX)
C
C           contient les composantes fixees de X
C
      DOUBLE PRECISION Y(NMAX),Z(NMAX)
C
C           vecteurs intermediaires
C
      LOGICAL OK
C
C           .TRUE. si on obtient un point admissible apres la resolution
C                  sans contraintes.
C
      DOUBLE PRECISION LAMBDA(NMAX)
C
C           multilpicateurs de lagrange
C
      INTEGER T
      DOUBLE PRECISION SINUS(NMAX),COSINUS(NMAX),MAX
      DOUBLE PRECISION GRAND,MCHEPS,PETIT,ZERO,UN
C
      DOUBLE PRECISION ALPHA
C
C           pas effectue le long de la direction de recherche
C
      DOUBLE PRECISION Q(NMAX)
C
C           solution du probleme sans contraintes
C
      DOUBLE PRECISION DDOT
      EXTERNAL DDOT
C
      INTEGER ITER
C
C           nombre d'iterations
C
      INTEGER CPU,CPUQR,MAJ
      INTEGER ISECONDIFF
      EXTERNAL ISECONDIFF
C
C           pour le calcul du temps CPU
C
C     =============
C     ! Programme !
C     =============
C
C     Calcul des caracteristiques de la machine
C     -----------------------------------------
C
      CALL DCMACH(MCHEPS,GRAND,PETIT,ZERO,UN)
```

```
C
C       Initialisations
C       ---------------
C
        CALL ISECOND(CPU)
        ITER=0
        DO 2 I=1,N
           P(I)=I
    2   CONTINUE
C
C       Nombre de variable libres
C
        FF=N
C
C       Nombre de variables actives
C
        BB=0
        DO 10 I=1,N
           ACT(I)=0
           X(I)=(L(I)+U(I))/2
   10   CONTINUE
        CPUQR=ISECONDIFF(CPU)
C
C       Decomposition QR de A
C
        CALL DQRDC(A,MMAX,M,N,QRAUX,JPVT,WORK,0)
C                    t
C       Calcul de Q b
C
        CALL DQRSL(A,MMAX,M,N,QRAUX,B,DUMMY1,QTB,DUMMY2,DUMMY3,DUMMY4,
        1          1000,INFO)
        CPUQR=ISECONDIFF(CPU)-CPUQR
C
C       Iterations
C       ----------
C
    1   ITER=ITER+1
C
        IF (ITER.GT.MAXITE) THEN
           WRITE(US,*)'Trop d''iterations !!'
           RETURN
        ENDIF
C
C       Calcul de la solution sans contraintes
C
        IF (BB.NE.0) THEN
C
C          Calcul de E(BB).x
C
           DO 20 J=1,BB
                XB(J)=X(P(J+FF))
   20      CONTINUE
```

```
C
C           Test sur le nombre de variables libres.
C
            IF (FF.EQ.0) THEN
               WRITE(US,*)'Toutes les variables sont actives'
               OK=.TRUE.
C
C              On calcule les multiplicateurs de Lagrange
C
               GOTO 55
            ENDIF
C
C           Calcul de S.XB
C
            DO 30 I=1,FF
                  Z(I)=DDOT(BB,A(I,FF+1),MMAX,XB,1)
   30       CONTINUE
         ENDIF
         CALL DCOPY(FF,QTB,1,Y,1)
         IF (BB.NE.0) CALL DAXPY(FF,-UN,Z,1,Y,1)
         CALL DTRSL(A,MMAX,FF,Y,01,INFO)
C
C        Calcul de z=E(FF).q
C
         DO 40 J=1,FF
            Z(P(J))=Y(J)
   40    CONTINUE
C
C        La solution sans contrainte est-elle admissible ?
C
         OK=.TRUE.
         I=1
   50    IF (OK .AND. I.LE.N) THEN
            IF (ACT(I).EQ.0) THEN
                  OK=(Z(I)-L(I).GT.MCHEPS .AND. U(I)-Z(I).GT.MCHEPS )
            ENDIF
            I=I+1
            GOTO 50
         ENDIF
C
C        Construction du nouvel itere.
C
   55    IF (OK) THEN
            DO 60 I=1,N
                  IF (ACT(I).EQ.0) THEN
                     X(I)=Z(I)
                  ENDIF
   60       CONTINUE
C
C           Test d'optimalite.
C
            IF (BB.EQ.0) THEN
                  GOTO 150
            ELSE
```

```
C
C                Calcul des multiplicateurs de Lagrange
C
                 DO 70 I=1,BB
                    Y(I)=QTB(I+FF)-DDOT(BB-I+1,A(I+FF,I+FF),MMAX
     1                    ,XB(I),1)
   70            CONTINUE
                 DO 80 I=1,BB
                    LAMBDA(P(I+FF))=DDOT(I,A(1+FF,I+FF),1,Y,1)
   80            CONTINUE
C
C                Recherche d'un candidat a etre desactive.
C
                 MAX=ZERO
                 T=0
                 DO 100 J=1,N
                    IF (ACT(J).NE.0) THEN
                          IF(ACT(J)*LAMBDA(J).GT.MAX) THEN
                             T=J
                             MAX=ACT(J)*LAMBDA(J)
                          ENDIF
                    ENDIF
  100            CONTINUE
C
C                Test d'optimalite.
C
                 IF (T.EQ.0) THEN
                    GOTO 150
                 ELSE
                    ACT(T)=0
                    BB=BB-1
                    FF=FF+1
C
C                   T n'est plus active
C                   On met la colonne T en FFieme place et on deplace les
C                   autre colonnes vers la gauche
C
                    DO 38632 J=1,N
                          IF (P(J).EQ.T) THEN
                             K=J
                             MAJ=ISECONDIFF(CPU)
                             CALL DCHEX(A,MMAX,N,FF,K,QTB,MMAX,1
     1                             ,COSINUS,SINUS,1)
                             CPUQR=CPUQR+ISECONDIFF(CPU)-MAJ
                             CALL PERM(N,NMAX,P,FF,K,'RIGHT')
C
C                            On recommence une nouvelle iteration
C
                             GOTO 1
                          ENDIF
38632               CONTINUE
                 ENDIF
              ENDIF
           ELSE
```

```
C
C           Calcul du pas maximum.
C
            ALPHA=GRAND
            DO 110 I=1,N
                  IF (ACT(I).EQ.0) THEN
                     IF (Z(I).LT.L(I)) THEN
                           ALPHA=MIN(ALPHA,(X(I)-L(I))
     1                                    /(X(I)-Z(I)))
                     ELSE IF (U(I).LT.Z(I)) THEN
                           ALPHA=MIN(ALPHA,(U(I)-X(I))
     1                              /(Z(I)-X(I)))
                     ENDIF
                  ENDIF
  110       CONTINUE
C
C           Mise a jour du nouvel itere.
C
            DO 120 I=1,N
                  IF (ACT(I).EQ.0) THEN
C
                     X(I)=X(I)+ALPHA*(Z(I)-X(I))
C
                     IF (DABS(X(I)-L(I)).LT.MCHEPS) THEN
                        X(I)=L(I)
                        ACT(I)=1
                     ELSE
                        IF (DABS(U(I)-X(I)).LT.MCHEPS) THEN
                           X(I)=U(I)
                           ACT(I)=-1
                        ELSE
                           GO TO 120
                        ENDIF
                     ENDIF
C
                     BB=BB+1
                     FF=FF-1
C
C                    La variable I devient active
C
                     DO 61283 J=1,N
                           IF (P(J).EQ.I) THEN
                              K=J
                              CALL PERM(N,NMAX,P,K,FF+1,'LEFT')
                              MAJ=ISECONDIFF(CPU)
                              CALL DCHEX(A,MMAX,N,K,FF+1,QTB,MMAX,
     1                                 1,COSINUS,SINUS,2)
                              CPUQR=CPUQR+ISECONDIFF(CPU)-MAJ
C
C                             On examine la variable suivante
C
                              GOTO 120
                           ENDIF
61283                CONTINUE

                  ENDIF
  120       CONTINUE
            ENDIF
```

```
C
C       Nouvelle iteration.
C
        GOTO 1
C
C       Impressions finales.
C
  150   WRITE(US,155)
  155   FORMAT(' On est a la solution',/,
        1       ' ====================')
        CPU=ISECONDIFF(CPU)
        IF (N.LT.20) THEN
           WRITE(US,160)(I,X(I),I=1,N)
  160      FORMAT(' X(',I3,')=',1PD15.7)
        ENDIF
        WRITE(US,170)ITER
  170   FORMAT(' Nombre d''iterations :          ',I3)
        WRITE(US,180)BB
  180   FORMAT(' Nombre de variables actives :  ',I3)
        WRITE(US,190)CPU
  190   FORMAT(' Temps CPU :                    ',I7)
        WRITE(US,195)CPUQR
  195   FORMAT(' Temps CPU pour QR :            ',I7)
        RETURN
        END
```

```
      PROGRAM ESSAI
      ^^^^^^^^^^^^^
C
C
C     Ce programme permet de resoudre un probleme des
C     moindres carres avec contraintes de bornes en utilisant,
C     soit la nouvelle methode (routine MCARRE),
C     soit la methode de Bjorck.
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire    1988
C
C     ===============
C     | Parametres |
C     ===============
C
      INTEGER NMAX
      PARAMETER (NMAX=1000)
C
C           dimension maximale de X
C
      INTEGER MMAX
      PARAMETER (MMAX=1000)
C
C           dimension maximale de B
C
      INTEGER NZEROMAX
      PARAMETER (NZEROMAX=31000)
C
C           nombre maximal d'elements non nuls dans la matrice creuse A
C
C     ==============
C     | Variables |
C     ==============
C
      DOUBLE PRECISION A(NZEROMAX)
C
C           vecteur contenant les elements non nuls de la matrice A
C           stockes colonne par colonne
C
      INTEGER INDLIGNE(NZEROMAX)
C
C           vecteur contenant les indices lignes des elements non nuls
C           de A colonne par colonne
C
      INTEGER DEBCOL(NMAX+1)
C
C           vecteur contenant les pointeurs vers les premiers elements
C           non nuls des colonnes successives dans INDLIGNE et A.
C           DEBCOL(i) contient la position dans A et INDLIGNE du premier
C           element de la ligne no i. DEBCOL(N+1) contient la longueur
C           effective des vecteurs A et DEBLIGNE
```

```
C     DOUBLE PRECISION B(MMAX)
C
C             vecteur contenant le membre de droite du probleme
C
C     INTEGER M
C
C             dimension effective de B et nombre de lignes de A
C
C     INTEGER N
C
C             dimension effective de X et nombre de colonnes de A
C
C     DOUBLE PRECISION AP(MMAX,NMAX)
C
C             matrice A stockee de la maniere habituelle
C
C     DOUBLE PRECISION L(NMAX),U(NMAX)
C
C             vecteurs contenant les bornes imposees au variables
C             L contient les bornes inferieures
C             U contient les bornes superieures
C
C     DOUBLE PRECISION UPPER,LOWER
C
C             valeur des bornes (ce sont les memes bornes pour toutes les
C             variables)
C
C     DOUBLE PRECISION X(NMAX)
C
C             solution du probleme (du moins on l'espere !)
C
C     DOUBLE PRECISION ATOL
C
C             estimation de l'erreur relative sur les donnees definissant
C             la matrice A
C
C     DOUBLE PRECISION BTOL
C
C             estimation de l'erreur relative sur les donnees definissant
C             le vecteur B
C
C     DOUBLE PRECISION CONLIM
C
C             borne superieure sur le conditionnement de A
C
C     DOUBLE PRECISION XNLIM
C
C             borne superieure sur la norme 2 de la solution X
C
C     INTEGER US
C
C             numero de l'unite de sortie des impressions
C
```

```
      DOUBLE PRECISION EPSILON
C
C           precision desiree
C
      INTEGER MAXITE
C
C           nombre maximum d'iterations
C
      INTEGER RESLSQR
C
C           impressions demandees pour le LSQR
C           Si < 0  pas de resultat
C           Si = 0  resultat a la premiere et la derniere iteration
C           Si > 0  resultat toutes les RESLSQR iterations
C
      CHARACTER NOMFIC*8
C
C           nom des fichier
C
      CHARACTER COMMENT*80
C
C           commentaires du fichier de lecture
C
      INTEGER IO
C
C           erreur d'entree/sortie
C
      INTEGER I,J
C
C           variables de boucle
C
      CHARACTER REP*1
C
C           reponse a (O/N) ou a (1./2.)
C
      DOUBLE PRECISION RESUL
C
C           valeur de la quadratique a la solution
C
      INTEGER CALEE(NMAX)
      DOUBLE PRECISION ES1(NMAX),ES2(NMAX),NORMB,DDOT
      EXTERNAL DDOT
C
C     ======================
C     | Routines utilisees |
C     ======================
C
C     Dans la librairie UTIL
C     ----------------------
C
C     INIT
```

```
C       Dans le fichier MCARRE
C       ----------------------
C
C       MCARRE
C
C       Dans le fichier BJORCK
C       ----------------------
C
C       BJORCK
C
C       Dans le fichier ROUTINES
C       ------------------------
C
C       TRANSFORM,QUADRAT
C
C       ====================================
C       ! Structure du fichier de donnees !
C       ====================================
C
C       (Exemple pour une matrice 2 x 2)
C
C       Dimensions du probleme(MxN)
C       2
C       2
C       Pointeurs vers le debut des colonnes
C       1
C       3
C       4
C       Elements non nuls de A
C       2.d00
C       4.d00
C       -1.d00
C       Indices des lignes
C       1
C       2
C       1
C       Vecteur B
C       1.d00
C       4.d00
C       Tolerance sur les donnees de A
C       1.d-17
C       Tolerance sur les donnees de B
C       1.d-17
C       Limite du conditionnement de A
C       1.d17
C       Limite sur la norme de X
C       1.d17
C       Precision desiree
C       1.d-14
C       Nombre maximum d'iterations
C       10
```

```
C
C       ==============
C       ! Programme !
C       ==============
C
        CALL INIT
C
C       Ouverture du fichier de lecture
C
C       Repeter
C
   40   CONTINUE
           WRITE(*,45)
   45      FORMAT(' Donnez le nom du fichier de lecture (max 8 car.)')
           READ(*,'(A8)',ERR=50,IOSTAT=IO)NOMFIC
   50      IF (IO.NE.0) THEN
                   WRITE(*,*)' Erreur d''entree/sortie'
                   GOTO 40
           ENDIF
           US=70
           OPEN(UNIT=US,ERR=60,IOSTAT=IO,FILE=NOMFIC,
        1       STATUS='OLD')
   60      IF (IO.NE.0)THEN
                   WRITE(*,*)'Erreur d''ouverture du fichier'
                   GOTO 40
           ENDIF
C
C       Jusqu'a ce qu'il n'y ait plus d'erreur
C
C
C       Dimensions du probleme
C
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)M
        READ(US,*,ERR=50)N
C
C       Vecteur DEBCOL
C
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)(DEBCOL(I),I=1,N+1)
C
C       Elements non nuls de A
C
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)(A(I),I=1,DEBCOL(N+1)-1)
C
C       Indices des lignes
C
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)(INDLIGNE(I),I=1,DEBCOL(N+1)-1)
```

```
C       Vecteur B
C
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)(B(I),I=1,M)
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50,ATOL
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)BTOL
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)CONLIM
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)XNLIM
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)EPSILON
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)MAXITE
        CLOSE(US)
        WRITE(*,*)'Borne inferieure :'
        READ(*,*)LOWER
        WRITE(*,*)'Borne superieure :'
        READ(*,*)UPPER
        DO 237 I=1,N
           L(I)=LOWER
           U(I)=UPPER
  237   CONTINUE
        NORMB=DDOT(M,B,1,B,1)
        WRITE(*,345)NORMB
  345   FORMAT(' Norme de B : ',1PD15.7)
  212   WRITE(*,213)
  213   FORMAT(/,' Quelle methode desirez-vous utiliser ?',/
        1        ,5X,'1. Methode de Bjorck',/
        2        ,5X,'2. Nouvelle methode')
        READ (*,'(A1)',ERR=212) REP
        IF ((REP.NE.'1').AND.(REP.NE.'2')) GOTO 212

 1000   WRITE(*,*)'Donnez le nom du fichier de sortie '
        READ(*,'(A8)',ERR=1000)NOMFIC
        OPEN(UNIT=US,FILE=NOMFIC,ERR=1000,STATUS='NEW',
        1       CARRIAGECONTROL='LIST')
        IF (REP.EQ.'2') THEN
           CALL MCARRE(A,INDLIGNE,DEBCOL,B,M,N,L,U,X,ATOL,BTOL,CONLIM,
        1                XNLIM,US,1,EPSILON,MAXITE,0,1)
           CALL QUADRAT(X,A,INDLIGNE,DEBCOL,F,M,N,RESUL)
           WRITE(US,723)RESUL
  723      FORMAT(' Valeur de la quadratique a la solution :',1PD15.7)
        ELSE
           CALL TRANSFORM(M,N,A,INDLIGNE,DEBCOL,AP)
           CALL BJORCK(AP,M,N,X,B,L,U,MAXITE,US,A,INDLIGNE,DEBCOL)
        ENDIF
        CLOSE(US)
 1010   WRITE(*,*)'Une autre execution ?'
        READ(*,'(A1)',ERR=1010)REP
```

```
      IF ((REP.EQ.'O').OR.(REP.EQ.'o')) THEN
1020     WRITE(*,*)'Meme probleme mais autre methode ?'
         READ(*,'(A1)',ERR=1020)REP
         IF ((REP.EQ.'O').OR.(REP.EQ.'o')) THEN
              GOTO 212
         ELSE
              GOTO 40
         ENDIF
      ENDIF
      END
```

```
      PROGRAM CONDITION
      ~~~~~~~~~~~~~~~~~
C
C
C     Ce programme a pour but de tester les routines MCARRE et
C     BJORCK pour la resolution de moindres carres lineaires avec
C     contraintes de bornes, dans le cas de matrices mal conditionnees
C
C     Pour cela, je vais rajouter a une matrice A quelconque, une
C     colonne qui vaudra la colonne(i)+EPS, EPS variant de 0 a 1
C     et i etant le numero d'une des colonnes de A
C
C     J'aurai ainsi une colonne presque egale a une autre
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire    1988
C
C     ===============
C     | Parametres |
C     ===============
C
      INTEGER NMAX
      PARAMETER (NMAX=1000)
C
C           dimension maximale de X
C
      INTEGER MMAX
      PARAMETER (MMAX=1000)
C
C           dimension maximale de B
C
      INTEGER NZEROMAX
      PARAMETER (NZEROMAX=31000)
C
C           nombre maximal d'elements non nuls dans la matrice creuse A
C
      DOUBLE PRECISION ZERO,UN
      PARAMETER (ZERO=0.D00,UN=1.D00)
C
C     =============
C     | Fonction |
C     =============
C
      DOUBLE PRECISION DNRM2
      EXTERNAL DNRM2
C
C     ==============
C     | Variables |
C     ==============
C
      DOUBLE PRECISION A(NZEROMAX)
C
C           vecteur contenant les elements non nuls de la matrice A
C           stockes colonne par colonne
```

```
      INTEGER INDLIGNE(NZEROMAX)
C
C           vecteur contenant les indices lignes des elements non nuls
C           de A colonne par colonne
C
      INTEGER DEBCOL(NMAX+1)
C
C           vecteur contenant les pointeurs vers les premiers elements
C           non nuls des colonnes successives dans INDLIGNE et A.
C           DEBCOL(i) contient la position dans A et INDLIGNE du premier
C           element de la ligne no i. DEBCOL(N+1) contient la longueur
C           effective des vecteurs A et DEBLIGNE
C
      DOUBLE PRECISION B(MMAX)
C
C           vecteur contenant le membre de droite du probleme
C
      INTEGER M
C
C           dimension effective de B et nombre de lignes de A
C
      INTEGER N
C
C           dimension effective de X et nombre de colonnes de A
C
      DOUBLE PRECISION L(NMAX),U(NMAX)
C
C           vecteurs contenant les bornes imposees au variables
C           L contient les bornes inferieures
C           U contient les bornes superieures
C
      DOUBLE PRECISION X(NMAX)
C
C           solution du probleme (du moins on l'espere !)
C
      DOUBLE PRECISION ATOL
C
C           estimation de l'erreur relative sur les donnees definissant
C           la matrice A
C
      DOUBLE PRECISION BTOL
C
C           estimation de l'erreur relative sur les donnees definissant
C           le vecteur B
C
      DOUBLE PRECISION CONLIM
C
C           borne superieure sur le conditionnement de A
C
      DOUBLE PRECISION XNLIM
C
C           borne superieure sur la norme 2 de la solution X
C
      INTEGER US
C
C           numero de l'unite de sortie des impressions
```

```
      INTEGER IMPR
C
C           impressions de la routine MCARRE
C           Si < 0  pas de resultat
C           Si = 0  resultat a la premiere et la derniere iteration
C           Si > 0  resultat toutes les IMPR iterations
C
      DOUBLE PRECISION EPSILON
C
C           precision desiree
C
      INTEGER MAXITE
C
C           nombre maximum d'iterations
C
      INTEGER RESLSQR
C
C           impressions demandees pour le LSQR
C           Si < 0  pas de resultat
C           Si = 0  resultat a la premiere et la derniere iteration
C           Si > 0  resultat toutes les RESLSQR iterations
C
      CHARACTER NOMFIC*8
C
C           nom des fichier
C
      CHARACTER COMMENT*80
C
C           commentaires du fichier de lecture
C
      INTEGER IO
C
C           erreur d'entree/sortie
C
      INTEGER I,J,K
C
C           variables de boucle
C
      CHARACTER REP*1
C
C           reponse a (O/N)
C
      DOUBLE PRECISION AI(MMAX)
C
C           contient la i eme colonne de la matrice
C
      DOUBLE PRECISION EPS
C
C           constante ajoutee a la colonne i
C
      DOUBLE PRECISION EP
C
C           constante ajoutee a la colonne i (normalisee)
```

```
      DOUBLE PRECISION LOWER,UPPER
C
C           bornes
C
      DOUBLE PRECISION AP(MMAX,NMAX)
C
C           matrice stockee sous forme pleine
C
C     =============
C     ! Programme !
C     =============
C
      CALL INIT
C
C     Ouverture du fichier de lecture
C
C     Repeter
C
   40 CONTINUE
         WRITE(*,45)
   45    FORMAT(' Donnez le nom du fichier de lecture (max 8 car.)')
         READ(*,'(A8)',ERR=50,IOSTAT=IO)NOMFIC
   50    IF (IO.NE.0) THEN
                  WRITE(*,*)' Erreur d''entree/sortie'
                  STOP
                  GOTO 40
C        ENDIF
         US=70
         OPEN(UNIT=US,ERR=60,IOSTAT=IO,FILE=NOMFIC,
     1        STATUS='OLD')
   60    IF (IO.NE.0)THEN
                  WRITE(*,*)'Erreur d''ouverture du fichier'
                  STOP
                  GOTO 40
C        ENDIF
C
C     Jusqu'a ce qu'il n'y ait plus d'erreur
C
C
C     Dimensions du probleme
C
      READ(US,'(A80)',ERR=50,IOSTAT=IO)COMMENT
      READ(US,*,ERR=50,IOSTAT=IO)M
      READ(US,*,ERR=50,IOSTAT=IO)N
C
C     Vecteur DEBCOL
C
      READ(US,'(A80)',ERR=50,IOSTAT=IO)COMMENT
      READ(US,*,ERR=50,IOSTAT=IO)(DEBCOL(I),I=1,N+1)
C
C     Elements non nuls de A
C
      READ(US,'(A80)',ERR=50,IOSTAT=IO)COMMENT
      READ(US,*,ERR=50,IOSTAT=IO)(A(I),I=1,DEBCOL(N+1)-1)
```

```
      Indices des lignes

      READ(US,'(A80)',ERR=50,IOSTAT=IO)COMMENT
      READ(US,*,ERR=50,IOSTAT=IO)(INDLIGNE(I),I=1,DEBCOL(N+1)-1)

      Vecteur B

      READ(US,'(A80)',ERR=50,IOSTAT=IO)COMMENT
      READ(US,*,ERR=50,IOSTAT=IO)(B(I),I=1,M)

      READ(US,'(A80)',ERR=50,IOSTAT=IO)COMMENT
      READ(US,*,ERR=50,IOSTAT=IO)ATOL
      READ(US,'(A80)',ERR=50,IOSTAT=IO)COMMENT
      READ(US,*,ERR=50,IOSTAT=IO)BTOL
      READ(US,'(A80)',ERR=50,IOSTAT=IO)COMMENT
      READ(US,*,ERR=50,IOSTAT=IO)CONLIM
      READ(US,'(A80)',ERR=50,IOSTAT=IO)COMMENT
      READ(US,*,ERR=50,IOSTAT=IO)XNLIM
      READ(US,'(A80)',ERR=50,IOSTAT=IO)COMMENT
      READ(US,*,ERR=50,IOSTAT=IO)EPSILON
      READ(US,'(A80)',ERR=50,IOSTAT=IO)COMMENT
      READ(US,*,ERR=50,IOSTAT=IO)MAXITE
      CLOSE(US)
C
   44 WRITE(*,*)'Bornes :'
      READ(*,*,ERR=50,IOSTAT=IO)LOWER,UPPER
      IF (UPPER.LE.LOWER) GOTO 44
      DO 24 I=1,N
         L(I)=LOWER
         U(I)=UPPER
   24 CONTINUE
C
C     Fichier de sortie
C
 1000 WRITE(*,*)'Donnez le nom du fichier de sortie '
      READ(*,'(A8)',ERR=50,IOSTAT=IO)NOMFIC
      US=69
      OPEN(UNIT=US,FILE=NOMFIC,ERR=50,IOSTAT=IO,STATUS='NEW',
     1        CARRIAGECONTROL='LIST')
C
C     On prend la premiere colonne et on lui ajoute EPS
C
      WRITE(*,*)'Epsilon :'
      READ(*,*)EPS
         CALL COLONNE(A,INDLIGNE,DEBCOL,AI,M,1)
C
C        On divise EPS par la norme de la premiere colonne
C
         EP=EPS/DNRM2(M,AI,1)
C
C        On perturbe la colonne
C
         DO 23 I=1,M
               AI(I)=AI(I)+EP
   23    CONTINUE
```

```
C
C       On ajoute la colonne perturbee a la matrice A
C
C
        J=DEBCOL(N+1)
        DO 27 I=1,M
              IF (AI(I).NE.ZERO) THEN
                 A(J)=AI(I)
                 INDLIGNE(J)=I
                 J=J+1
              ENDIF
   27   CONTINUE
        N=N+1
        DEBCOL(N+1)=J
        L(N)=LOWER
        U(N)=UPPER
        WRITE(US,543)EPS
  543   FORMAT('Epsilon =',1PD15.7)
  212   WRITE(*,213)
  213   FORMAT(/,' Quelle methode desirez-vous utiliser ?',/
     1         ,5X,'1. Methode de Bjorck',/
     2         ,5X,'2. Nouvelle methode')
        READ (*,'(A1)',ERR=212) REP
        IF ((REP.NE.'1').AND.(REP.NE.'2')) THEN
              WRITE(US,*)'Erreur dans le fichier de donnees'
              STOP
        ENDIF
        IF (REP.EQ.'2') THEN
           CALL MCARRE(A,INDLIGNE,DEBCOL,B,M,N,L,U,X,ATOL,BTOL,
     1        CONLIM,XNLIM,US,0,EPSILON,1000,-1,1)
        ELSE
              CALL TRANSFORM(M,N,A,INDLIGNE,DEBCOL,AP)
              CALL BJORCK(AP,M,N,X,B,L,U,MAXITE,US)
        ENDIF
C
C       On supprime la derniere colonne de la matrice
C
C
        CALL SUPPRCOL(A,INDLIGNE,DEBCOL,N,N,ZERO)
      CLOSE(US)
 1010 WRITE(*,*)'Une autre execution ?'
      READ(*,'(A1)',ERR=1010)REP
      IF ((REP.EQ.'O').OR.(REP.EQ.'o')) GOTO 40
      END
```

```
      PROGRAM COMPLEMENT
      ~~~~~~~~~~~~~~~~~~
C
C
C     Ce programme va tester le comportement de l'algorithme MCARRE et
C     celui de l'algorithme de Bjorck lorsque les conditions de
C     complementarite stricte de sont pas verifiees
C
C     En fait, on va resoudre un probleme sans contraintes de bornes.
C     Ensuite, on va placer une contrainte de borne juste a la valeur de la
C     solution.
C     Pour etudier l'evolution du phenomene, on va reculer petit a petit
C     cette borne
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire  1988
C
C     ================
C     | Parametres |
C     ================
C
      INTEGER NMAX
      PARAMETER (NMAX=1000)
C
C           dimension maximale de X
C
      INTEGER MMAX
      PARAMETER (MMAX=1000)
C
C           dimension maximale de B
C
      INTEGER NZEROMAX
      PARAMETER (NZEROMAX=31000)
C
C           nombre maximal d'elements non nuls dans la matrice creuse A
C
C     ==============
C     | Variables |
C     ==============
C
      DOUBLE PRECISION A(NZEROMAX)
C
C           vecteur contenant les elements non nuls de la matrice A
C           stockes colonne par colonne
C
      INTEGER INDLIGNE(NZEROMAX)
C
C           vecteur contenant les indices lignes des elements non nuls
C           de A colonne par colonne
```

```
C
C     INTEGER DEBCOL(NMAX+1)
C
C             vecteur contenant les pointeurs vers les premiers elements
C             non nuls des colonnes successives dans INDLIGNE et A.
C             DEBCOL(i) contient la position dans A et INDLIGNE du premier
C             element de la ligne no i. DEBCOL(N+1) contient la longueur
C             effective des vecteurs A et DEBLIGNE
C
C     DOUBLE PRECISION AP(MMAX,NMAX)
C
C             matrice A stockee de la maniere habituelle
C
C     DOUBLE PRECISION B(MMAX)
C
C             vecteur contenant le membre de droite du probleme
C
C     INTEGER M
C
C             dimension effective de B et nombre de lignes de A
C
C     INTEGER N
C
C             dimension effective de X et nombre de colonnes de A
C
C     DOUBLE PRECISION L(NMAX),U(NMAX)
C
C             vecteurs contenant les bornes imposees au variables
C             L contient les bornes inferieures
C             U contient les bornes superieures
C
C     DOUBLE PRECISION X(NMAX)
C
C             solution du probleme (du moins on l'espere !)
C
C     DOUBLE PRECISION ATOL
C
C             estimation de l'erreur relative sur les donnees definissant
C             la matrice A
C
C     DOUBLE PRECISION BTOL
C
C             estimation de l'erreur relative sur les donnees definissant
C             le vecteur B
C
C     DOUBLE PRECISION CONLIM
C
C             borne superieure sur le conditionnement de A
C
C     DOUBLE PRECISION XNLIM
C
C             borne superieure sur la norme 2 de la solution X
C
C     INTEGER US
C
C             numero de l'unite de sortie des impressions
```

```
      DOUBLE PRECISION EPSILON
C
C           precision desiree
C
      INTEGER MAXITE
C
C           nombre maximum d'iterations
C
      INTEGER RESLSQR
C
C           impressions demandees pour le LSQR
C           Si < 0  pas de resultat
C           Si = 0  resultat a la premiere et la derniere iteration
C           Si > 0  resultat toutes les RESLSQR iterations
C
      DOUBLE PRECISION RECUL
C
C           longueur dont on recule la borne par rapport a la solution
C            sans contraintes
C
      CHARACTER NOMFIC*8
C
C           nom des fichier
C
      CHARACTER COMMENT*80
C
C           commentaires du fichier de lecture
C
      INTEGER IO
C
C           erreur d'entree/sortie
C
      INTEGER I,J
C
C           variables de boucle
C
      CHARACTER REP*1
C
C           reponse a (O/N)
C
      INTEGER STRICT
C
C           nombre de variables pour lesquelles a condition ne sera
C           pas verifiee
C
      DOUBLE PRECISION RESUL
C
C           residu final
C
C     ==============
C     ! Programme !
C     ==============
C
      CALL INIT
```

```
C
C       Ouverture du fichier de lecture
C
C       Repeter
C
40      CONTINUE
           WRITE(*,45)
45         FORMAT(' Donnez le nom du fichier de lecture (max 8 car.)')
           READ(*,'(A8)',ERR=50,IOSTAT=IO)NOMFIC
50         IF (IO.NE.0) THEN
                   WRITE(*,*)' Erreur d''entree/sortie'
                   GOTO 40
           ENDIF
           US=70
           OPEN(UNIT=US,ERR=60,IOSTAT=IO,FILE=NOMFIC,
     1          STATUS='OLD')
60         IF (IO.NE.0)THEN
                   WRITE(*,*)'Erreur d''ouverture du fichier'
                   GOTO 40
           ENDIF
C
C       Jusqu'a ce qu'il n'y ait plus d'erreur
C
C
C
C       Dimensions du probleme
C
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)M
        READ(US,*,ERR=50)N
C
C       Vecteur DEBCOL
C
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)(DEBCOL(I),I=1,N+1)
C
C       Elements non nuls de A
C
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)(A(I),I=1,DEBCOL(N+1)-1)
C
C       Indices des lignes
C
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)(INDLIGNE(I),I=1,DEBCOL(N+1)-1)
C
C       Vecteur B
C
        READ(US,'(A80)',ERR=50)COMMENT
        READ(US,*,ERR=50)(B(I),I=1,M)
```

```
      READ(US,'(A80)',ERR=50)COMMENT
      READ(US,*,ERR=50)ATOL
      READ(US,'(A80)',ERR=50)COMMENT
      READ(US,*,ERR=50)BTOL
      READ(US,'(A80)',ERR=50)COMMENT
      READ(US,*,ERR=50)CONLIM
      READ(US,'(A80)',ERR=50)COMMENT
      READ(US,*,ERR=50)XNLIM
      READ(US,'(A80)',ERR=50)COMMENT
      READ(US,*,ERR=50)EPSILON
      READ(US,'(A80)',ERR=50)COMMENT
      READ(US,*,ERR=50)MAXITE
      CLOSE(US)
 1000 WRITE(*,*)'Donnez le nom du fichier de sortie '
      READ(*,'(A8)',ERR=1000)NOMFIC
      OPEN(UNIT=US,FILE=NOMFIC,ERR=1000,STATUS='NEW',
     1       CARRIAGECONTROL='LIST')
      WRITE(*,*)'Donnez le nombre de bornes a determiner :'
      READ(*,*)STRICT
      IF (STRICT.GT.N) STOP
      WRITE(*,*)'De combien faut-il reculer les bornes ?'
      READ(*,*)RECUL
  212 WRITE(*,213)
  213 FORMAT(/,' Quelle methode desirez-vous utiliser ?',/
     1       ,5X,'1. Methode de Bjorck',/
     2       ,5X,'2. Nouvelle methode')
      READ (*,'(A1)',ERR=212) REP
      IF ((REP.NE.'1').AND.(REP.NE.'2')) THEN
         WRITE(US,*)'Erreur dans le fichier de donnees'
         STOP
      ENDIF
C
C     On resoud sans contraintes
C
      DO 467 I=1,N
         L(I)=-1.D5
         U(I)=1.D5
  467 CONTINUE
      CALL TRANSFORM(M,N,A,INDLIGNE,DEBCOL,AP)
      CALL MCARRE(A,INDLIGNE,DEBCOL,B,M,N,L,U,X,ATOL,BTOL,CONLIM,
     1       XNLIM,US,0,EPSILON,MAXITE,-1,0)
C
C     On met les bornes superieures aux valeurs des STRICT premieres
C     variables
C
      DO 70 I=1,STRICT
         U(I)=X(I)-RECUL
   70 CONTINUE
      WRITE(US,37)RECUL
   37 FORMAT(/,/,' On est avant la solution de ',1PD15.7)
      IF (REP.EQ.'2') THEN
         CALL MCARRE(A,INDLIGNE,DEBCOL,B,M,N,L,U,X,ATOL,BTOL,
     1       CONLIM,XNLIM,US,0,EPSILON,MAXITE,-1,0)
      ELSE
         CALL BJORCK(AP,M,N,X,B,L,U,MAXITE,US)
      ENDIF
      CALL QUADRAT(X,A,INDLIGNE,DEBCOL,B,M,N,RESUL)
      WRITE(US,4567)RESUL
 4567 FORMAT('Residu : ',1PD15.7)
      CLOSE(US)
      END
```

```
      subroutine clsqr2( m, n, a, x, u, indrow, colbeg,
     +                   iter, maxit, rnorm, atol, btol,
     +                   acond, conlim, xnorm, xnlim,
     +                   istop, w, ipfreq, ipdevc,
     +                   anorm, arnorm,lo,sup,calee)
```

```
*     Cette routine resoud le probleme des moindres carres lineaires
*     en utilisant la methode iterative LSQR.
*     Le probleme sera resolu uniquement pour certaines variables,
*     appelees variables libres et determinees par le vecteur 'calee'

*     Elle est prevue pour etre utilisee avec une matrice creuse non
*     symetrique qui est stockee en exploitant la structure creuse.

*     La routine donne egalement une estimation du conditionnement
*     du probleme et une estimation de la norme de la solution

*     Elle tient compte de contraintes de bornes sur les variables,
*     en ce sens qu'elle s'arrete des qu'elle sort du domaine
*     admissible


*     Parametres
*     ----------

      double precision one,zero
      parameter (one = 1.d00,zero = 0.d00)

*     Arguments de la routine
*     -----------------------

      integer m,n

*       input  : dimensions de la matrice A
*       output : inchange

      double precision a(1)

*       input  : vecteur contenant les elements non nuls
*                de la matrice definissant le probleme,
*                stockes colonne par colonne
*       output : inchange

      double precision x(1)

*       input  : sans importance (vecteur de longueur au moins n)
*       output : la meilleure solution trouvee

      double precision u(1)

*       input  : un vecteur de longueur au moins egale a m,
*                contenant le terme independant du probleme
*       output : sans importance (mais transforme)
```

```
      integer indrow(1)

*      input  : vecteur contenant les indices des lignes
*               des elements non nuls de a, stockes
*               colonne par colonne
*      output : inchange

      integer colbeg(1)

*      input  : vecteur contenant les pointeurs vers les premiers
*               elements des colonnes successives dans 'indrow' et
*               'a' : colbeg(i) contient la position dans 'a' et
*               'indrow' du premier element de la i ieme colonne.
*               colbeg(n+1) est defini comme la longueur de 'a' et
*               'indrow', plus 1.
*      output : inchange

      integer iter

*      input  : sans importance
*      output : le nombre d'iterations effectuees par la routines

      integer maxit

*      input  : nombre maximum d'iterations autorise
*      output : inchange

      double precision rnorm

*      input  : sans importance
*      output : estimation de la norme du residu final

      double precision atol

*      input  : estimation de l'erreur relative dans
*               les donnees de 'a'
*      output : inchange

      double precision btol

*      input  : estimation de l'erreur relative dans
*               les donnees de 'u'
*      output : inchang?

      double precision acond

*      input  : sans importance
*      output : estimation du conditionnement de 'a'

      double precision conlim

*      input  : borne superieure sur le conditionnement de 'a'.
*               La routine s'arrete si cette borne est depassee
*      output : inchange
```

```
      double precision xnorm

*      input  : sans importance
*      output : estimation de la norme du point 'x'

      double precision xnlim

*      input  : borne superieure sur la norme de 'x'.
*               La routine s'arrete si cette borne est depassee
*      output : inchange

      integer istop

*      input  : sans importance
*      output : raison de l'arret de la routine :
*               1. on a trouve une solution exacte du systeme
*               2. on a atteint le nombre maximum d'iterations
*               3. on a trouve une solution de moindre carre
*               4. le conditionnement de 'a' depasse la borne
*                        permise
*               5. la norme de 'x' a depasse la borne permise
*               6. 'x' se trouve hors du domaine admissible

      double precision w(1)

*      input  : sans importance (vecteur de longueur au moins 2*n)
*      output : sans importance (c'est un vecteur de travail)

      integer ipfreq

*      input  : frequence des sorties produites par la routine
*               si ipfreq < 0 alors pas de sorties du tout
*               si ipfreq = 0 alors sortie a la premiere et a
*                             la derniere iteration uniquement
*               si ipfreq > 0 alors sortie chaque ipfreq iteration
*      output : inchange

      integer ipdevc

*      input  : numero logique de sortie
*      output : inchange

      double precision anorm

*      input  : sans importance
*      output : estimation de la norme de Frobenius de 'a'

      double precision arnorm

*      input  : sans importance
*      output : estimation de la norme du gradient du residu

      double precision lo(1),sup(1)

*      input  : bornes definissant le domaine admissible,
*               duquel la routine ne peut pas sortir
*      output : inchange
```

```
      integer calee(1)

c     input  : vecteur de longueur au moins egale a n precisant
c              quelles variables il faut prendre en compte
c              calee(i)=0 si on doit tenir compte de la variable i
c     output : inchange


c     Routines utilisees
c     ------------------

c     Dans la librairie ANLIB  : destvl,dnrmlz
c     Dans le fichier ROUTINES : ax,psnrzx,pscpyx,pssclx
c     Dans la librairie BLAS   : dscal

c     Programmation
c     -------------

c     Ph. L. Toint
c     Adaptations : M. Bierlaire

c     Variables de la routine
c     -----------------------

      integer           i,lw,itemp
      double precision alfa,bbnorm,beta,bnorm,cs,cs2,ctol,ddnorm
      double precision delta,gamma,gambar,phi,phibar,rho,rhobar
      double precision rhs,rtol,sn,sn2,t,tau,test1,test2,test3
      double precision theta,t1,t2,t3,xxnorm,z,zbar,beta2,xlim

c     Programme
c     ---------

c     Initialisations

      xlim   = abs(xnlim)
      if(conlim.gt.one)then
        ctol = one/conlim
      else
        ctol = zero
      end if
      anorm  = zero
      acond  = zero
      bbnorm = zero
      ddnorm = zero
      xnorm  = zero
      xxnorm = zero
      cs2    = -one
      sn2    = zero
      z      = zero
      iter   = 0
      istop  = 0
      lw = n + 1
```

```
*    Mise a zero de x et w

       call dsetvl(n,x,1,zero)
       call dsetvl(n,w(1),1,zero)

*    Initialisation des premiers vecteurs pour la bidiagonalisation

       call dnrmlz(m,u,1,beta,one)
       call ax(m,n,a,indrow,colbeg,u,w(1),calee,.true.,.true.)
       call psnrzx(n,w,alfa,one,calee)
       call pscpyx(n,w(1),w(lw),calee)

       rhobar = alfa
       phibar = beta
       bnorm  = beta
       rnorm  = beta
       arnorm = alfa*beta

*    Impression du message de depart

       if(ipfreq.ge.0)then
         write(ipdevc,1000)
1000     format(//' ***** Entering clsqr1 routine *****')
         write(ipdevc,2000)iter,rnorm,arnorm
2000     format(/'iteration ',i4,' : approximate residual norm = ',
     +     pd14.7,/'                  gradient norm             = ',
     +     pd14.7,/)
       end if

*    Test d'arret pour le point de depart

       if(arnorm.le.zero)then
         istop = 1
         if(ipfreq.ge.0)write(ipdevc,3000)
3000     format(/' ***** Exiting  clsqr1 routine *****')
         return
       end if

       if(maxit.le.0)then
         istop = 2
         if(ipfreq.ge.0)write(ipdevc,3000)
         return
       end if

       if(xlim+one.le.one)then
         istop = 5
         if(ipfreq.ge.0)write(ipdevc,3000)
         return
       end if

*    Boucle principale

       do 100 iter=1,maxit
```

```
*    Pas suivant pour la bidiagonalisation

     call dscal (m,(-alfa),u,1)
     call ax(m,n,a,indrow,colbeg,w(1),u,calee,.false.,.true.)
     call dnrmlz(m,u,1,beta,one)
     beta2  = beta*beta
     bbnorm = bbnorm + alfa*alfa + beta2
     call pssclx (n,(-beta),w,calee)
     call ax(m,n,a,indrow,colbeg,u,w(1),calee,.true.,.true.)
     call psnrzx(n,w,alfa,one,calee)

*    Rotation plane pour eliminer les elements sous diagonaux

     rho    = sqrt(rhobar*rhobar + beta2)
     cs     = rhobar/rho
     sn     =   beta/rho
     theta  =  sn*alfa
     rhobar = -cs*alfa
     phi    = cs*phibar
     phibar = sn*phibar
     tau    = sn*phi

*    Mise a jour de x et w

     t1     =    phi/rho
     t2     = -theta/rho
     t3     =    one/rho
     do 200 i=1,n
     if (calee(i).eq.0) then
       itemp    = n + i
       t        = w(itemp)
       x(i)     = x(i) + t1*t
       if (x(i).lt.lo(i) .or. x(i).gt.sup(i)) istop=6
       w(itemp) = w(i) + t2*t
       ddnorm   = ddnorm + (t3*t)**2
     endif
200  continue

     if(istop.gt.0)then
       if(ipfreq.ge.0)write(ipdevc,3000)
       return
     end if

*    Rotation plane pour eliminer les elements sur-diagonaux
*    theta et estimation de la norme de x

     delta  =  sn2*rho
     gambar = -cs2*rho
     rhs    = phi - delta*z
     zbar   = rhs/gambar
     xnorm  = sqrt(xxnorm + zbar*zbar)

*    Calcul des valeurs pour les tests

     anorm  = sqrt(bbnorm)
     acond  = anorm*sqrt(ddnorm)
     rnorm  = abs(phibar)
     arnorm = alfa*abs(tau)
```

```
      test1  = rnorm/bnorm
      if (rnorm.ne.0) then
         test2  = arnorm/(anorm*rnorm)
      else
         test2 = 1.d12
      endif
      test3  = one/acond
      t2     = anorm*xnorm/bnorm
      t1     = test1/(one + t2)
      rtol   = btol + atol*t2

      t3 = one + test3
      t2 = one + test2
      t1 = one + t1

*     Tests de convergence

*     1) Solution exacte

      if(test1.le.rtol.or.t1.le.one)istop=1

*     2) Systeme trop mal conditionne

      if(test3.le.ctol.or.t3.le.one)istop = 4

*     3) Solution des moindres carres

      if(test2.le.atol.or.t2.le.one)istop = 3

*     4) Trop d'iterations

      if(iter.ge.maxit)istop = 2

*     5) Solution trop grande

      if(xnorm.ge.xlim)istop = 5

*     Impressions

      if(ipfreq.ge.0)then
        if(ipfreq.gt.0)then
          itemp = mod(iter,ipfreq)
          if(itemp.eq.0)write(ipdevc,2000)iter,rnorm,arnorm
        else
          if(istop.gt.0)
     1       write(ipdevc,2000)iter,rnorm,arnorm
        end if
      end if

*     Sortie ?

      if(istop.gt.0)then
        if(ipfreq.ge.0)write(ipdevc,3000)
        return
      end if
```

```
*     Boucle

          gamma   = sqrt(gambar*gambar + theta*theta)
          cs2     = gambar/gamma
          sn2     = theta/gamma
          z       = rhs/gamma
          xxnorm  = xxnorm + z*z

100       continue

          return

          end
```

```
      SUBROUTINE GRAD(M,N,X,A,G,INDLIGNE,DEBCOL)
      ~~~~~~~~~~~~~~~~
C
C
C     Cette routine evalue le gradient
C
C                t     t
C               A Ax-A b
C
C     de la fonction quadratique au point x, A etant stocke sous forme
C     creuse.
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire    1988
C
C     ================
C     ! Parametres !
C     ================
C
      INTEGER NMAX
      PARAMETER (NMAX=1000)
C
C           dimension maximale de X
C
      INTEGER MMAX
      PARAMETER (MMAX=1000)
C
C           dimension maximale de B
C
      INTEGER NZEROMAX
      PARAMETER (NZEROMAX=31000)
C
C           nombre maximal d'elements non nuls dans la matrice creuse A
C
      DOUBLE PRECISION ZERO
      PARAMETER (ZERO=0.D00)
C
C     ================
C     ! Arguments !
C     ================
C
      INTEGER M
C
C        INPUT :
C             dimension effective de B et nombre de lignes de A
C        OUTPUT :
C             inchange
C
      INTEGER N
C
C        INPUT :
C             dimension effective de X et nombre de colonnes de A
C        OUTPUT :
C             inchange
```

```
      DOUBLE PRECISION X(NMAX)

C        INPUT :
C             point ou on calcule le gradient
C        OUTPUT :
C             inchange

      DOUBLE PRECISION A(NZEROMAX)

C        INPUT :
C             vecteur contenant les elements non nuls de la matrice A
C             stockes colonne par colonne
C        OUTPUT :
C             inchange

      DOUBLE PRECISION ATB(NMAX)

C        INPUT :          t
C             valeur de - A B
C        OUTPUT :
C             inchange

      DOUBLE PRECISION G(NMAX)

C        INPUT :
C             sans importance
C        OUTPUT :
C             valeur du gradient en X

      INTEGER INDLIGNE(NZEROMAX)

C        INPUT :
C             vecteur contenant les indices lignes des elements non nuls
C             de A colonne par colonne
C        OUTPUT :
C             inchange

      INTEGER DEBCOL(NMAX+1)

C        INPUT :
C             vecteur contenant les pointeurs vers les premiers elements
C             non nuls des colonnes successives dans INDLIGNE et A.
C             DEBCOL(i) contient la position dans A et INDLIGNE du premier
C             element de la ligne no i. DEBCOL(N+1) contient la longueur
C             effective des vecteurs A et DEBLIGNE
C        OUTPUT :
C             inchange

      DOUBLE PRECISION MZERO(MMAX)

C        INPUT :              m
C             vecteur nul de R
C        OUTPUT :
C             inchange
```

```
      DOUBLE PRECISION NZERO(NMAX)

C        INPUT :                  n
C             vecteur nul de R
C        OUTPUT :
C             inchange

C     ============================
C     ! Variables du programme !
C     ============================

      DOUBLE PRECISION MY(MMAX)

C             vecteur intermediaire

C     ==========================
C     ! Routines utilisees !
C     ==========================

C     Dans la librairie ANLIB :
C     -------------------------

C     DSETVL

C     Dans le fichier ROUTINES
C     ------------------------

C     RSCSP

C     ===============
C     ! Programme !
C     ===============

C     Calcul de Ax
C     ------------

      MY=0

      CALL DSETVL(M,MY,1,ZERO)

      MY=Ax

      CALL RSCSP(M,N,A,X,MY,.FALSE.,.TRUE.,INDLIGNE,DEBCOL)
                        t     t
C     Calcul de A Ax-A b
C     ------------------

      CALL RSCSP(M,N,A,MY,G,.TRUE.,.TRUE.,INDLIGNE,DEBCOL)
      RETURN
      END
```

```
C
      SUBROUTINE ADM(X,DIM,L,U,OK,ZERO)
C     ~~~~~~~~~~~~~~~~
C
C     Cette routine crouve un point admissible.
C     Si le vecteur nul est admissible, elle retourne celui-ci.
C     Sinon, elle met toutes les variables pour lesquelles 0 n'est pas
C     admissible, a leur borne superieure.
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire  1988
C
C     ================
C     | Parametres |
C     ================
C
      INTEGER NMAX
      PARAMETER (NMAX=1000)
C
C           dimension maximale de X
C
C     ================
C     | Arguments |
C     ================
C
      DOUBLE PRECISION X(NMAX)
C
C        INPUT :
C             sans importance
C        OUTPUT :
C             vecteur admissible
C
      INTEGER DIM
C
C        INPUT :
C             dimension effective de X,L et U
C        OUTPUT :
C             inchange
C
      DOUBLE PRECISION L(NMAX),U(NMAX)
C
C        INPUT :
C             bornes sur les variables, definissant le domaine admissible
C        OUTPUT :
C             inchanges
C
      LOGICAL OK
C
C        INPUT :
C             sans importance
C        OUTPUT :
C             .TRUE. si 0 est admissible
```

```
      DOUBLE PRECISION ZERO

          INPUT :
               valeur 0.d00
          OUTPUT :
               inchange

      ================================
      ! Variables de la routine !
      ================================

      INTEGER I

               Variable de boucle

      ==============
      ! Programme !
      ==============

      OK=.TRUE.
      DO 10 I=1,DIM
               IF ((L(I).GT.ZERO).OR.(U(I).LT.ZERO)) THEN
                  OK=.FALSE.
                  X(I)=L(I)
               ELSE
                  X(I)=ZERO
               ENDIF
10    CONTINUE
      RETURN
      END
```

```
C
C
      SUBROUTINE RECHERCHE(X,N,CALEE,GRAD,D,L,U,EPS,P)
C     ~~~~~~~~~~~~~~~~~~~~~~~~
C
C
C     Cette routine determine les composantes actives d'un vecteur X,
C     relativement aux contraintes de borne L et U.
C
C     Elle donne egalement une premiere direction de recherche pour le
C     calcul du point de Cauchy generalise.
C
C     Elle indique aussi la dimension du sous espace des variables
C     libres : P.
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire ,  1988
C
C     ===============
C     | Parametres |
C     ===============
C
      INTEGER NMAX
      PARAMETER (NMAX=1000)
C
C           dimension maximale de X,L et U
C
      DOUBLE PRECISION ZERO
      PARAMETER (ZERO=0.D00)
C
C     ===============
C     | Arguments |
C     ===============
C
      DOUBLE PRECISION X(NMAX)
C
C        INPUT :
C              vecteur dont on doit trouver les composantes actives
C        OUTPUT :
C              inchange
C
       INTEGER N
C
C        INPUT :
C              dimension effective de X,L et U,...
C        OUTPUT :
C              inchange
```

```
      INTEGER CALEE(NMAX)

C
C        INPUT :
C              sans importance
C        OUTPUT :
C
C              CALEE(i) = 0 si X(i) est une variable libre
C              CALEE(i) = 1 si X(i) atteint sa borne inferieure
C              CALEE(i) = 3 si X(i) atteint sa borne superieure
C

      DOUBLE PRECISION GRAD(1)

C
C        INPUT :
C              gradient au point X
C        OUTPUT :
C              inchange
C

      DOUBLE PRECISION D(1)

C
C        INPUT :
C              sans importance
C        OUTPUT :
C              direction de recherche
C

      DOUBLE PRECISION L(NMAX),U(NMAX)

C
C        INPUT :
C              vecteurs contenant les bornes imposees au variables
C              L contient les bornes inferieures
C              U contient les bornes superieures
C        OUTPUT :
C              inchange
C

      DOUBLE PRECISION EPS

C              precision de la machine
C

      INTEGER P

C
C         INPUT :
C               sans importance
C         OUTPUT :
C               dimension du sous-espace des variables libres
C
C     ============================
C     | Variables de la routine |
C     ============================
C
      INTEGER I

C               variable de boucle
C
```

```
C      ===============
C      | Programme |
C      ===============
C
C
       P=0
       DO 10 I=1,N
          D(I)=-GRAD(I)
          IF (ABS(X(I)-L(I)).LT.EPS) THEN
                CALEE(I)=1
                P=P+1
                IF (GRAD(I).GT.ZERO) D(I)=ZERO
          ELSE IF (ABS(X(I)-U(I)).LT.EPS) THEN
                CALEE(I)=3
                P=P+1
                IF (GRAD(I).LT.ZERO) D(I)=ZERO
          ELSE
                CALEE(I)=0
          ENDIF
   10  CONTINUE
       END
C
C
```

```
C
C
      SUBROUTINE COLONNE(A,INDLIGNE,DEBCOL,AI,M,I)
      ~~~~~~~~~~~~~~~~~~~~
C
C     Cette routine va copier la I ieme colonne de A dans le vecteur AI
C     (A est stockee sous forme creuse)
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire    1988
C
C     ===============
C     | Parametres |
C     ===============
C
      INTEGER NMAX
      PARAMETER (NMAX=1000)
C
C           dimension maximale des lignes de A
C
      INTEGER MMAX
      PARAMETER (MMAX=1000)
C
C           dimension maximale des colonnes de A
C
      INTEGER NZEROMAX
      PARAMETER (NZEROMAX=31000)
C
C           nombre maximal d'elements non nuls dans la matrice creuse A
C
C     ===============
C     | Arguments |
C     ===============
C
      DOUBLE PRECISION A(NZEROMAX)
C
C        INPUT :
C              vecteur contenant les elements non nuls de la matrice A
C              stockes colonne par colonne
C        OUTPUT :
C              inchange
C
      INTEGER INDLIGNE(NZEROMAX)
C
C        INPUT :
C              vecteur contenant les indices lignes des elements non nuls
C              de A colonne par colonne
C        OUTPUT :
C              inchange
```

```
      INTEGER DEBCOL(NMAX+1)
C
C        INPUT :
C             vecteur contenant les pointeurs vers les premiers elements
C             non nuls des colonnes successives dans INDLIGNE et A.
C             DEBCOL(i) contient la position dans A et INDLIGNE du premier
C             element de la ligne no i. DEBCOL(N+1) contient la longueur
C             effective des vecteurs A et DEBLIGNE
C        OUTPUT :
C             inchange
C
      DOUBLE PRECISION AI(MMAX)
C
C        INPUT :
C             sans importance
C        OUTPUT :
C             contient la I ieme colonne de A
C
      INTEGER M
C
C        INPUT :
C             longueur effective des colonnes de A
C        OUTPUT :
C             inchange
C
      INTEGER I
C
C        INPUT :
C             numero de la colonne a copier
C        OUTPUT :
C             inchange
C
C     ==============================
C     | Variables de la routine |
C     ==============================
C
      INTEGER J,K
C
C             variables de boucle.
C
C     ===============
C     | Programme |
C     ===============
C
      J=1
      K=0
C
C     Tant que J <= M ...
C
   10 IF (J.LE.M) THEN
         IF (J.NE.INDLIGNE(DEBCOL(I)+K)) THEN
            AI(J)=0.
         ELSE
            AI(J)=A(DEBCOL(I)+K)
            K=K+1
         ENDIF
         J=J+1
         GOTO 10
      ENDIF
C     Fin Tant que
C
      END
C
C
```

```
      SUBROUTINE SUPPRCOL(A,INDLIGNE,DEBCOL,I,N,ZERO)
      ~~~~~~~~~~~~~~~~~~~~~~~

C     Cette routine va supprimer la colonne I de la matrice creuse A
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire    1988
C
C     ================
C     | Parametres |
C     ================
C
      INTEGER NMAX
      PARAMETER (NMAX=1000)
C
C           dimension maximale de X
C
      INTEGER MMAX
      PARAMETER (MMAX=1000)
C
C           dimension maximale de B
C
      INTEGER NZEROMAX
      PARAMETER (NZEROMAX=31000)
C
C           nombre maximal d'elements non nuls dans la matrice creuse A
C
C     ===============
C     | Arguments |
C     ===============
C
C           DOUBLE PRECISION A(NZEROMAX)
C
C        INPUT :
C              vecteur contenant les elements non nuls de la matrice A
C              stockes colonne par colonne
C        OUTPUT :
C              idem pour la matrice dont on a enleve la Iieme colonne
C
C     INTEGER INDLIGNE(NZEROMAX)
C
C        INPUT :
C              vecteur contenant les indices lignes des elements non nuls
C              de A colonne par colonne
C        OUTPUT :
C              idem pour la matrice dont on a enleve la Iieme colonne
```

```
      INTEGER DEBCOL(NMAX+1)
C
C        INPUT :
C             vecteur contenant les pointeurs vers les premiers elements
C             non nuls des colonnes successives dans INDLIGNE et A.
C             DEBCOL(i) contient la position dans A et INDLIGNE du premier
C             element de la ligne no i. DEBCOL(N+1) contient la longueur
C             effective des vecteurs A et DEBLIGNE
C        OUTPUT :
C             idem pour la matrice dont on a enleve la Iieme colonne
C
      INTEGER I
C
C        INPUT :
C             Numero de la colonne a supprimer
C        OUTPUT :
C             inchange
C
      INTEGER N
C
C        INPUT :
C             nombre de colonnes de la matrice A avant la suppression
C        OUTPUT :
C             nombre de colonnes apres la suppression (i.e. N-1)
C
      DOUBLE PRECISION ZERO
C
C        INPUT :
C             contient la valeur 0.d00
C        OUTPUT :
C             inchange
C
C     ==============================
C     | Variables de la routine |
C     ==============================
C
      INTEGER NONZERO
C
C             Nombre d'elements non-nuls dans la colonne I
C
      INTEGER J
C
C             Indice de boucle
C     ================
C     | Programme |
C     ================
C
      NONZERO=DEBCOL(I+1)-DEBCOL(I)
      DO 10 J=DEBCOL(I),DEBCOL(N+1)-NONZERO-1
             A(J)=A(J+NONZERO)
             INDLIGNE(J)=INDLIGNE(J+NONZERO)
10    CONTINUE
      DO 15 J=DEBCOL(N+1)-NONZERO,DEBCOL(N+1)-1
             A(J)=ZERO
             INDLIGNE(J)=0
15    CONTINUE
      DO 20 J=I,N
             DEBCOL(J)=DEBCOL(J+1)-NONZERO
20    CONTINUE
      DEBCOL(N+1)=0
      N=N-1
      RETURN
      END
```

```
C
C
      LOGICAL FUNCTION OPTIMAL(X,GRAD,N,PRECISION,L,U,ECRIRE,US)
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
C
C     Cette fonction verifie si le point X est une solution optimale
C     du probleme en calculant la norme du gradient projete
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire     1988
C
      DOUBLE PRECISION ZERO
      PARAMETER (ZERO=0.D00)
C
C     ==============
C     | Arguments |
C     ==============
C
      DOUBLE PRECISION X(1)
C
C        INPUT :
C              candidat pour la solution
C        OUTPUT :
C              inchange
C
      DOUBLE PRECISION GRAD(1)
C
C        INPUT :
C              gradient au point X
C        OUTPUT :
C              inchange
C
      INTEGER N
C
C        INPUT :
C              dimension des vecteurs X,GRAD,L,U
C        OUTPUT :
C              inchange
C
      DOUBLE PRECISION PRECISION
C
C        INPUT :
C              precision exigee
C        OUTPUT :
C              inchange
C
      DOUBLE PRECISION L(1),U(1)
C
C        INPUT :
C              vecteurs de bornes
C        OUTPUT :
C              inchanges
```

```
      LOGICAL ECRIRE

         INPUT :
              .true. si on desire des impressions
         OUTPUT :
              inchange

      INTEGER US

         INPUT :
              numero de l'unite logique de sortie
         OUTPUT :
              inchange

      ============================
      | Variables du programme |
      ============================

      INTEGER I

              indice de boucle

      DOUBLE PRECISION XX

              contient X(i) - GRAD(i)

      DOUBLE PRECISION GPNORM

              norme du gradient projete

      =============
      | Programme |
      =============

      GPNORM=ZERO
      DO 10 I=1,N
         XX=X(I)-GRAD(I)
         IF (XX.GT.U(I)) THEN
              GPNORM=GPNORM+(U(I)-X(I))**2
         ELSE IF (XX.LT.L(I)) THEN
              GPNORM=GPNORM+(L(I)-X(I))**2
         ELSE
              GPNORM=GPNORM+GRAD(I)**2
         ENDIF
   10 CONTINUE
      GPNORM=GPNORM**(0.5)
      IF (ECRIRE) THEN
         WRITE(US,612)GPNORM
  612    FORMAT('Norme du gradient projete : ',1PD15.7)
      ENDIF
      OPTIMAL=(GPNORM.LT.PRECISION)
      RETURN
      END
```

```
C
C
      SUBROUTINE AX(M,N,A,INDLIGNE,DEBCOL,X,U,CALEE,TRANSPOSE,ADDSUB)
      ~~~~~~~~~~~~~~

C     Cette routine effectue un produit matrice vecteur en
C     tenant compte de la structure creuse de A. De plus,
C     les variables pour lesquelles CALEE(i) <> 0 ne sont
C     pas prises en compte.
C     (i.e.  les x(i) si TRANSPOSE = .false.
C            les u(i) si TRANSPOSE = .true. )
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire    1988
C
C     ==============
C     | Arguments |
C     ==============
C
      INTEGER M
C
C        INPUT :
C              nombre de lignes de la matrice A
C              dimension du vecteur U si TRANSPOSE = .false.
C              dimension du vecteur X si TRANSPOSE = .true.
C        OUTPUT :
C              inchange
C
      INTEGER N
C
C        INPUT :
C              nombre de colonnes de la matrice A
C              dimension du vecteur X si TRANSPOSE = .false.
C              dimension du vecteur U si TRANSPOSE = .true.
C        OUTPUT :
C              inchange
C
      DOUBLE PRECISION A(1)
C
C        INPUT :
C              contient les elements non nuls de A
C        OUTPUT :
C              inchange
C
      INTEGER INDLIGNE(1)
C
C        INPUT :
C              contient les indices des lignes des elements non nuls de A
C              INDLIGNE(i) est le numero de la ligne de l'element non nul A(i
C        OUTPUT :
C              inchange
```

```
      INTEGER DEBCOL(1)

C
C        INPUT :
C              vecteur de pointeurs vers le debut de chaque colonne
C              DEBCOL(i) contient l'emplacement du premier element non nul
C                 de la colonne i de A
C              DEBCOL(N+1) contient 1 + (le nombre d'element non nuls de A)
C        OUTPUT :
C              inchange
C
      DOUBLE PRECISION X(1)
C                                                              t
C        INPUT :
C              vecteur qui va etre premultiplie par A ou A
C              il doit etre de dimension N si TRANSPOSE = .false.
C                           de dimension M si TRANSPOSE = .true.
C        OUTPUT :
C              inchange
C
      DOUBLE PRECISION U(1)
C
C        INPUT :
C              vecteur de dimension M si TRANSPOSE = .false.
C              vecteur de dimension N si TRANSPOSE = .true.
C        OUTPUT :
C              u <-- u + Ax     si ADDSUB = .true. et TRANSPOSE = .false.
C
C              u <-- u - Ax     si ADDSUB = .false. et TRANSPOSE = .false.
C                        t
C              u <-- u + A x     si ADDSUB = .true. et TRANSPOSE = .true.
C                        t
C              u <-- u - A x     si ADDSUB = .false. et TRANSPOSE = .true.
C
      INTEGER CALEE(1)
C
C        INPUT :
C              CALEE(i) = 0 si la i ieme variable est "libre"
C              CALEE(i) > 0 sinon
C        OUTPUT :
C              inchange
C
      LOGICAL TRANSPOSE
C
C        INPUT :
C              .false. si on veut effectuer le produit Ax
C                                                      t
C              .true. si on veut effectuer le produit A x
C
C        OUTPUT :
C              inchange
C
      LOGICAL ADDSUB
C
C        INPUT :
C              .true. si on veut ajouter le produit a U
C              .false. si on veut retrancher le produit de U
C        OUTPUT :
C              inchange
C
```

```
      ================================
      ! Variables de la routine !
      ================================

      INTEGER I,J

              variables de boucle

      LOGICAL FIXEE

              .true. si une variable est fixee

      ==============
      ! Programme !
      ==============

      IF (.NOT.TRANSPOSE) THEN
         DO 10 I=1,N
              IF (CALEE(I).EQ.0) THEN
                 DO 20 J=DEBCOL(I),DEBCOL(I+1)-1
                       IF (ADDSUB) THEN
                          U(INDLIGNE(J))=U(INDLIGNE(J))+A(J)*X(I)
                       ELSE
                          U(INDLIGNE(J))=U(INDLIGNE(J))-A(J)*X(I)
                       ENDIF
                 CONTINUE
20               ENDIF
10       CONTINUE
      ELSE
         DO 30 I=1,N
              IF(CALEE(I).EQ.0) THEN
                 IF (ADDSUB) THEN
                       DO 40 J=DEBCOL(I),DEBCOL(I+1)-1
                          U(I)=U(I)+A(J)*X(INDLIGNE(J))
                       CONTINUE
40                ELSE
                       DO 50 J=DEBCOL(I),DEBCOL(I+1)-1
                          U(I)=U(I)-A(J)*X(INDLIGNE(J))
                       CONTINUE
50                ENDIF
              ENDIF
30       CONTINUE
      ENDIF
      RETURN
      END
```

```
      SUBROUTINE PSSCLX(N,SCALF,X,XSTATE)

*****************************************************************
*                                                               *
*    This routine multiplies the non fixed components of X      *
*      by the factor SCALF                                      *
*                                                               *
*****************************************************************
```

Programming : Ph. Toint, D. Tuyttens

Parameters

N (int)

input : the number of components of X
output : unmodified

X (dble,i=1,N)

input : the vector of which one wishes to multiply the non fixed part
output : the non fixed components of X are multiplied by the factor SCALF

SCALF (dble)

input : scalar factor to multiply with the non fixed part of X
output : unmodified

XSTATE (int ,i=1,N)

input : the status vector for the vector X. If XSTATE(i)>0 then the i-th component of X is fixed
output : unmodified

```
      INTEGER           N,XSTATE(n)
      DOUBLE PRECISION  X(N),SCALF

      INTEGER           I


      DO 10 I=1,N
        IF(XSTATE(I).LE.0)X(I)=X(I)*SCALF
   10 CONTINUE
      RETURN
      END
```

```
      FUNCTION PSNR2X(N,X,XSTATE)

C
C
C     ****************************************************************
C     *                                                              *
C     *    This routine computes the euclidean norm of the non       *
C     *      fixed part of X                                         *
C     *                                                              *
C     ****************************************************************
C
C
C  Programming : Ph. Toint, D. Tuyttens
C  ------------
C
C  Parameters
C  ----------
C
C     N       (int )
C
C       input  : the number of components of X
C       output : unmodified
C
C
C     X       (dble,i=1,N)
C
C       input  : the vector for which one wishes to compute the
C                norm of the non fixed part
C       output : unmodified
C
C
C     PSNR2X  (dble)
C
C       input  : no assumed value
C       output : the euclidean norm of the non fixed part of X
C
C
C     XSTATE (int ,i=1,N)
C
C       input  : the status vector for the vector X. If
C                XSTATE(i)>0 then the i-th component of X is fixed
C       output : unmodified
C
C
      INTEGER           N,XSTAtE(n)
      DOUBLE PRECISION  X(N),PSNR2X
      DOUBLE PRECISION  VALEUR
C
      INTEGER           I
C
C
      VALEUR=0.0
      DO 10 I=1,N
        IF(XSTATE(I).LE.0)VALEUR=VALEUR+X(I)*X(I)
10    CONTINUE
      PSNR2X=SQRT(VALEUR)
      RETURN
      END
C
```

```
      SUBROUTINE PSNRZX(N,X,XNORM,NWNORM,XSTATE)

C
C
C     ****************************************************************
C     *                                                              *
C     *    This routine computes the euclidean norm of the non fixed *
C     *      part of X and returns it in XNORM. If XNORM is nonzero, *
C     *      The non fixed part of X is scaled to NWNORM             *
C     *                                                              *
C     ****************************************************************
C
C
C  Programming : Ph. Toint, D. Tuyttens
C  ------------
C
C  Parameters
C  ----------
C
C    N       (int )
C
C      input  : the dimension of the vector X
C      output : unmodified
C
C
C    X       (dble)
C
C      input  : a vector of double precision numbers
C      output : the non fixed components of X have been scaled
C               to NWNORM
C
C
C    XNORM   (dble)
C
C      input  : no assumed value
C      output : the euclidean norm of the non fixed part of X
C
C
C    NWNORM  (dble)
C
C      input  : the desired norm of the scaled X
C      output : unmodified
C
C
C    XSTATE  (int ,i=1,N)
C
C      input  : the status vector for the vector X. If
C               XSTATE(i)>0 then the i-th component of X is fixed
C      output : unmodified
C
C
      INTEGER           N,XSTATE(n)
      DOUBLE PRECISION  X(N),XNORM,NWNORM
      DOUBLE PRECISION  PSNR2X
C
      XNORM=PSNR2X(N,X,XSTATE)
      IF(XNORM.GT.0.0)CALL PSSCLX(N,(NWNORM/XNORM),X,XSTATE)
      RETURN
      END
C
```

```
C
      SUBROUTINE PSCPYX(N,X,Y,XSTATE)
C
C
C     ****************************************************************
C     *                                                              *
C     *   This routine copies the non fixed components of X in the   *
C     *     corresponding components of Y                            *
C     *                                                              *
C     ****************************************************************
C
C
C  Programming : Ph. Toint, D. Tuyttens
C  ------------
C
C  Parameters
C  ----------
C
C     N       (int )
C
C       input  : the number of components of X
C       output : unmodified
C
C
C     X       (dble,i=1,N)
C
C       input  : the vector from which one wishes to copy the
C                non fixed part
C       output : unmodified
C
C
C     Y       (dble,i=1,N)
C
C       input  : no assumed value
C       output : the non fixed part of Y is now equal to the
C                non fixed part of X
C
C
C     XSTATE (int ,i=1,N)
C
C       input  : the status vector for the vector X. If
C                XSTATE(i)>0 then the i-th component of X is fixed
C       output : unmodified
C
C
      INTEGER           N,XSTATE(n)
      DOUBLE PRECISION  X(N),Y(N)
C
      INTEGER           I
C
C
      DO 10 I=1,N
        IF(XSTATE(I).LE.0)Y(I)=X(I)
10    CONTINUE
      RETURN
      END
C
```

```
      SUBROUTINE PSNRMZ(N,X,XNORM,NWNORM)

C     ******************************************************************
C     *                                                                *
C     *    This routine computes the euclidean norm of the X           *
C     *      and returns it in XNORM. If XNORM is nonzero,             *
C     *      X is scaled to NWNORM                                     *
C     *                                                                *
C     ******************************************************************
C
C
C  Programming : Ph. Toint, D. Tuyttens
C  ------------
C
C  Parameters
C  ----------
C
C     N       (int )
C
C       input  : the dimension of the vector X
C       output : unmodified
C
C
C     X       (dble)
C
C       input  : a vector of double precision numbers
C       output : the components of X have been scaled
C                to NWNORM
C
C
C     XNORM   (dble)
C
C       input  : no assumed value
C       output : the euclidean norm of X
C
C
C     NWNORM  (dble)
C
C       input  : the desired norm of the scaled X
C       output : unmodified
C
C
      INTEGER           N
      DOUBLE PRECISION  X(N),XNORM,NWNORM
      DOUBLE PRECISION  PSNRM2
C
C
      XNORM= PSNRM2(N,X)
      IF(XNORM.GT.0.0)CALL PSSCAL(N,(NWNORM/XNORM),X)
      RETURN
      END
C
```

```
      SUBROUTINE PSSCAL(N,SCALF,X)

C*************************************************************************
C*                                                                       *
C*    This routine multiplies the components of X                        *
C*      by the factor SCALF                                              *
C*    (It can be replaced by BLAS S(D)SCAL with INCX=1)                  *
C*                                                                       *
C*************************************************************************

C Programming : Ph. Toint, D. Tuyttens
C ------------

C Parameters
C ----------

C   N       (int )
C
C     input  : the number of components of X
C     output : unmodified
C
C
C   X       (dble,i=1,N)
C
C     input  : the vector of which one wishes to multiply
C     output : the components of X are multiplied by SCALF
C
C
C   SCALF   (dble)
C
C     input  : scalar factor to multiply with X
C     output : unmodified
C
C
      INTEGER           N
      DOUBLE PRECISION  X(N),SCALF
C
      INTEGER           I
C
C
      DO 10 I=1,N
        X(I)=X(I)*SCALF
10    CONTINUE
      RETURN
      END
C
C
```

```
C
      FUNCTION PSNRM2(N,X)
C
C
C     ******************************************************************
C     *                                                                *
C     *    This routine computes the euclidean norm of X               *
C     *    (It can be replaced by BLAS S(D)NRM2 with INCX=INCY=1)      *
C     *                                                                *
C     ******************************************************************
C
C    Programming : Ph. Toint, D. Tuyttens
C    ------------
C
C    Parameters
C    ----------
C
C      N       (int )
C
C        input  : the number of components of X
C        output : unmodified
C
C
C      X       (dble,i=1,N)
C
C        input  : the vector of which one wishes to compute the
C                 norm
C        output : unmodified
C
C
C      PSNRM2 (dble)
C
C        input  : no assumed value
C        output : the euclidean norm of X
C
C
C
      INTEGER           N
      DOUBLE PRECISION  X(N),PSNRM2
C
      INTEGER           I
C
C
      PSNRM2=0.0
      DO 10 I=1,N
        PSNRM2=PSNRM2+X(I)**2
10    CONTINUE
      PSNRM2=SQRT(PSNRM2)
      RETURN
      END
C
C
```

```
C
C
      SUBROUTINE CASSURE(D,XX,L,U,MCEPS,GRAND,DELT,N,CALEE,TOUT)
C     ~~~~~~~~~~~~~~~~~~~~~~~~~~
C
C     Cette routine va determiner le premier point de rencontre d'une
C     direction avec le bord de la region admissible (dans le sous-espace
C     determine par CALEE
C     En fait, elle va determiner le pas qu'il faut faire le long de
C     cette direction pour arriver au bord de la region.
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire  1988
C
C     ================
C     | Parametres |
C     ================
C
      INTEGER NMAX
      PARAMETER (NMAX=1000)
C
C     ================
C     | Arguments |
C     ================
C
      DOUBLE PRECISION D(1)
C
C         INPUT :
C              direction de recherche
C         OUTPUT :
C              inchange
C
      DOUBLE PRECISION XX(1)
C
C         INPUT :
C              depart de la recherche
C         OUTPUT :
C              inchange
C
      DOUBLE PRECISION L(1),U(1)
C
C         INPUT :
C              vecteurs de bornes definissant la region admissible
C         OUTPUT :
C              inchanges
C
      DOUBLE PRECISION MCEPS
C
C         INPUT :
C              precision machine
C         OUTPUT :
C              inchange
```

```
      DOUBLE PRECISION GRAND
C
C        INPUT :
C              limite d'overflow de la machine
C        OUTPUT :
C              inchange
C
      DOUBLE PRECISION DELT
C
C        INPUT :
C              sans importance
C        OUTPUT :
C              pas qu'il faut faire le long de D pour rencontrer les bornes
C
      INTEGER N
C
C        INPUT :
C              dimension des differents vecteurs
C        OUTPUT :
C              inchange
C
      INTEGER CALEE(1)
C
C        INPUT :
C              etat de l'activite des variables (voir routine MCARRE)
C        OUTPUT :
C              inchange
C
      LOGICAL TOUT
C
C        INPUT :
C              .TRUE. si on travaille dans tout l'espace
C        OUTPUT :
C              inchange
```

```
C     ==============================
C     | Variables du programme |
C     ==============================

      INTEGER I

C             variable de boucle

      DOUBLE PRECISION DELTA

C             contient les differents pas pour chacune des variables

      LOGICAL ACT

C             .TRUE. si une variable est active

C     ===============
C     | Programme |
C     ===============

      DELT=GRAND
      DO 20 I=1,N
         IF (CALEE(I).EQ.0 .OR. TOUT) THEN
            IF (D(I).GT.MCEPS) THEN
                  DELTA=(U(I)-XX(I))/D(I)
                  DELT=MIN(DELT,DELTA)
            ELSE IF (D(I).LT.-MCEPS) THEN
                  DELTA=(L(I)-XX(I))/D(I)
                  DELT=MIN(DELT,DELTA)
            ENDIF
         ENDIF
   20 CONTINUE
      RETURN
      END
C     ------------------------------------------------------------------
```

```
C
C
      SUBROUTINE QUADRAT(X,A,INDLIGNE,DEBCOL,B,M,N,RESUL)
      ~~~~~~~~~~~~~~~~~~~

C     Cette routine evalue la valeur de la quadratique :
C
C                              2
C              || Ax - b ||
C
C     ou A est une matrice creuse
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire  1988
C
C     ================
C     | Parametres |
C     ================
C
      INTEGER NMAX
      PARAMETER (NMAX=1000)
      DOUBLE PRECISION ZERO,UN
      PARAMETER (ZERO=0.D00,UN=1.D00)
C
C     ==============
C     | Fonctions |
C     ==============
C
      DOUBLE PRECISION DDOT
      EXTERNAL DDOT
C
C     ==============
C     | Arguments |
C     ==============
C
      DOUBLE PRECISION X(1)
C
C        INPUT :
C             point ou on doit evaluer la quadratique
C        OUTPUT :
C             inchange
C
      DOUBLE PRECISION A(1)
C
C        INPUT :
C             contient les elements non nuls de A
C        OUTPUT :
C             inchange
```

```
C
C     INTEGER INDLIGNE(1)
C
C        INPUT :
C             contient les indices des lignes des elements non nuls de A
C             INDLIGNE(i) est le numero de la ligne de l'element non nul A(i)
C        OUTPUT :
C             inchange
C
C     INTEGER DEBCOL(1)
C
C        INPUT :
C             vecteur de pointeurs vers le debut de chaque colonne
C             DEBCOL(i) contient l'emplacement du premier element non nul
C                de la colonne i de A
C             DEBCOL(N+1) contient 1 + (le nombre d'element non nuls de A)
C        OUTPUT :
C             inchange
C
C     DOUBLE PRECISION B(1)
C
C        INPUT :
C             r.h.s. du probleme
C        OUTPUT :
C             inchange
C
C     INTEGER M
C
C        INPUT :
C             nombre de lignes de la matrice A
C        OUTPUT :
C             inchange
C
C     INTEGER N
C
C        INPUT :
C             dimension des differents vecteurs
C             nombre de colonnes de la matrice A
C        OUTPUT :
C             inchange
C
C     DOUBLE PRECISION RESUL
C
C        INPUT :
C             sans importance
C        OUTPUT :
C             valeur de la quadratique au point X
```

```
C     ========================
C     | Routines utilisees   |
C     ========================
C
C     Dans la librairie ANLIB :
C     -------------------------
C
C     DSETVL
C
C     Dans le fichier ROUTINES :
C     --------------------------
C
C     RSCSP
C
C     Dans la librairie BLAS :
C     ------------------------
C
C     DAXPY
C
C     ============================
C     | Variables du programme   |
C     ============================
C
      DOUBLE PRECISION Y(NMAX)

C              vecteur intermediaire

C     ==============
C     | Programme  |
C     ==============

      CALL DSETVL(M,Y,1,ZERO)

C     Calcul de Y = Ax

      CALL RSCSP(M,N,A,X,Y,.FALSE.,.TRUE.,INDLIGNE,DEBCOL)

C     Calcul de Ax - b

      CALL DAXPY(M,-UN,B,1,Y,1)

C     Calcul du resultat : < Ax - b , Ax - b > = || Ax - b ||2

      RESUL=DDOT(M,Y,1,Y,1)

      RETURN
      END
```

```
C
      SUBROUTINE TRANSFORM(M,N,AC,INDLIGNE,DEBCOL,AP)
      ~~~~~~~~~~~~~~~~~~~~~~~
C
C     Cette routine va transformer la structure creuse de A en
C     une structure pleine
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire , 1988
C
C     ================
C     | Parametres |
C     ================
C
      INTEGER NMAX,MMAX,NZEROMAX
      PARAMETER (NMAX=1000,MMAX=1000,NZEROMAX=31000)
C
C     ========================
C     | Routines utilisees |
C     ========================
C
C     Dans la librairie BLAS :
C     ------------------------
C
C     DCOPY
C
C     Dans le fichier ROUTINES :
C     --------------------------
C
C     COLONNE
C
C     ===============
C     | Arguments |
C     ===============
C
      INTEGER M,N
C
C         INPUT :
C              dimensions de la matrice A
C         OUTPUT :
C              inchanges
C
      DOUBLE PRECISION AC(NZEROMAX)
C
C         INPUT :
C              vecteur contenant les elements non nuls de la matrice A
C              stockes colonne par colonne
C         OUTPUT :
C              inchange
```

```
      INTEGER INDLIGNE(NZEROMAX)
C
C        INPUT :
C             vecteur contenant les indices lignes des elements non nuls
C             de A colonne par colonne
C        OUTPUT :
C             inchange
C
      INTEGER DEBCOL(NMAX+1)
C
C        INPUT :
C             vecteur contenant les pointeurs vers les premiers elements
C             non nuls des colonnes successives dans INDLIGNE et A.
C             DEBCOL(i) contient la position dans A et INDLIGNE du premier
C             element de la ligne no i. DEBCOL(N+1) contient la longueur
C             effective des vecteurs A et DEBLIGNE
C        OUTPUT :
C             inchange
C
      DOUBLE PRECISION AP(MMAX,NMAX)
C
C        INPUT :
C             sans importance
C        OUTPUT :
C             contient la matrice A
C
C     ==============================
C     | Variables du programme |
C     ==============================
C
      INTEGER I
C
C             variable de boucle
C
      DOUBLE PRECISION AI(MMAX)
C
C             contient la i ieme colonne de A
C
C     ===============
C     | Programme |
C     ===============
C
      DO 10 I=1,N
         CALL COLONNE(AC,INDLIGNE,DEBCOL,AI,M,I)
         CALL DCOPY(M,AI,1,AP(1,I),1)
   10 CONTINUE
      RETURN
      END
```

```
C
      SUBROUTINE PERM(N,NMAX,P,K,L,SENS)
      ~~~~~~~~~~~~~~~~~~
C
C     Cette routine opere une permutation circulaire dans le sens
C     prevu par SENS, sur le vecteur d'entiers P
C
C     Shift circulaire vers la droite :
C
C     1,..,k-1,k,k+1,....,l,l+1,..n
C
C     devient
C
C     1,..,k-1,l,k,k+1,..,l-1,l+1,..,n
C
C
C     Shift circulaire vers la gauche :
C
C     1,..,k-1,k,...,l-1,l,l+1,..,n
C
C     devient
C
C     1,..,k-1,k+1,..,l,k,l+1,..,n
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire  1988
C
C     ==============
C     | Arguments |
C     ==============
C
      INTEGER N
C
C             dimension effective du vecteur P
C
      INTEGER NMAX
C
C             espace memoire prevu pour P
C
      INTEGER P(1)
C
C             vecteur a permuter
C
      INTEGER K
C
C             premiere colonne de P a etre changee
C
      INTEGER L
C
C             derniere colonne de P a etre changee
C             L doit etre strictement plus grand que K
C
      CHARACTER SENS*5
C
C             specifie le sens de la permutation
```

```
C      ==============================
C      ! Variables du programme !
C      ==============================
C
       INTEGER KEEP,I
C
C      ==============
C      ! Programme !
C      ==============
C
       IF (SENS.EQ.'RIGHT') THEN
          KEEP=P(L)
          DO 10 I=L,K+1,-1
                P(I)=P(I-1)
10        CONTINUE
          P(K)=KEEP
       ELSE
          KEEP=P(K)
          DO 20 I=K,L-1
                P(I)=P(I+1)
20        CONTINUE
          P(L)=KEEP
       ENDIF
       RETURN
       END
```

```
C
C
      SUBROUTINE RSCSP(M,N,A,X,U,TRANSPOSE,ADDSUB,INDLIGNE,DEBCOL)
      ~~~~~~~~~~~~~~~
C
C
C     Cette routine effectue un produit matrice-vecteur ou la matrice
C     est stockee sous forme creuse
C
C     Programmation :
C     ---------------
C
C     Michel Bierlaire    1988
C
C     ==============
C     | Arguments |
C     ==============
C
      INTEGER M
C
C        INPUT :
C             nombre de lignes de la matrice A
C             dimension du vecteur U si TRANSPOSE = .false.
C             dimension du vecteur X si TRANSPOSE = .true.
C        OUTPUT :
C             inchange
C
      INTEGER N
C
C        INPUT :
C             nombre de colonnes de la matrice A
C             dimension du vecteur X si TRANSPOSE = .false.
C             dimension du vecteur U si TRANSPOSE = .true.
C        OUTPUT :
C             inchange
C
      DOUBLE PRECISION A(1)
C
C        INPUT :
C             contient les elements non nuls de A
C        OUTPUT :
C             inchange
C
      INTEGER INDLIGNE(1)
C
C        INPUT :
C             contient les indices des lignes des elements non nuls de A
C             INDLIGNE(i) est le numero de la ligne de l'element non nul A(i)
C        OUTPUT :
C             inchange
```

```
INTEGER DEBCOL(1)

    INPUT :
         vecteur de pointeurs vers le debut de chaque colonne
         DEBCOL(i) contient l'emplacement du premier element non nul
            de la colonne i de A
         DEBCOL(N+1) contient 1 + (le nombre d'element non nuls de A)
    OUTPUT :
         inchange

DOUBLE PRECISION X(1)

                                                          t
    INPUT :
         vecteur qui va etre premultiplie par A ou A
         il doit etre de dimension N si TRANSPOSE = .false.
                      de dimension M si TRANSPOSE = .true.
    OUTPUT :
         inchange

DOUBLE PRECISION U(1)

    INPUT :
         vecteur de dimension M si TRANSPOSE = .false.
         vecteur de dimension N si TRANSPOSE = .true.
    OUTPUT :
         u <-- u + Ax     si ADDSUB = .true. et TRANSPOSE = .false.

         u <-- u - Ax     si ADDSUB = .false. et TRANSPOSE = .false.
                    t
         u <-- u + A x     si ADDSUB = .true. et TRANSPOSE = .true.
                    t
         u <-- u - A x     si ADDSUB = .false. et TRANSPOSE = .true.

LOGICAL TRANSPOSE

    INPUT :
         .false. si on veut effectuer le produit Ax
                                                   t
         .true. si on veut effectuer le produit A x

    OUTPUT :
         inchange

LOGICAL ADDSUB

    INPUT :
         .true. si on veut ajouter le produit a U
         .false. si on veut retrancher le produit de U
    OUTPUT :
         inchange
```

```
C      ==============================
C      | Variables de la routine |
C      ==============================
C
       INTEGER I,J

C
C             variables de boucle
C
       LOGICAL FIXEE

C
C             .true. si une variable est fixee
C
C      ==============
C      | Programme |
C      ==============
C
       IF (.NOT.TRANSPOSE) THEN
          DO 10 I=1,N
                DO 20 J=DEBCOL(I),DEBCOL(I+1)-1
                      IF (ADDSUB) THEN
                         U(INDLIGNE(J))=U(INDLIGNE(J))+A(J)*X(I)
                      ELSE
                         U(INDLIGNE(J))=U(INDLIGNE(J))-A(J)*X(I)
                      ENDIF
20              CONTINUE
10        CONTINUE
       ELSE
          DO 30 I=1,N
                IF (ADDSUB) THEN
                      DO 40 J=DEBCOL(I),DEBCOL(I+1)-1
                         U(I)=U(I)+A(J)*X(INDLIGNE(J))
40                    CONTINUE
                ELSE
                      DO 50 J=DEBCOL(I),DEBCOL(I+1)-1
                         U(I)=U(I)-A(J)*X(INDLIGNE(J))
50                    CONTINUE
                ENDIF
30        CONTINUE
       ENDIF
       RETURN
       END
C
C
```